Deep Learning For Time Series Classification Using New Hand-Crafted Convolution Filters

Ali Ismail-Fawaz, Maxime Devanne, Jonathan Weber, Germain Forestier

Deep Learning For Time Series Classification Using New Hand-Crafted Convolution Filters

Ali Ismail-Fawaz
Université de Haute Alsace
IRIMAS
Mulhouse, France
ali-el-hadi.ismail-fawaz@uha.fr

Maxime Devanne
Université de Haute Alsace
IRIMAS
Mulhouse, France
maxime.devanne@uha.fr

Jonathan Weber
Université de Haute Alsace
IRIMAS
Mulhouse, France
jonathan.weber@uha.fr

Germain Forestier
Université de Haute Alsace
IRIMAS
Mulhouse, France
germain.forestier@uha.fr

***Abstract*—In recent years, there has been an increasing interest in Deep Learning models for time series classification. In this field, state-of-the-art architectures rely on convolution neural networks that learn one dimensional filters in order to capture patterns allowing to discriminate between the different classes. These filters are randomly initialized and modified throughout model training. In this paper, we explore the creation of hand-crafted (non learned) filters in order to capture specific patterns in a time series. We propose a set of filters whose values are fixed and not modified during the training step. Our goal with these filters is to capture specific patterns in a time series (increase, decrease, peaks) and study the relevance of adding such filters to existing architectures ranging from simple architecture (Fully Convolutional Network (FNC)) to state-of-the-art architecture (InceptionTime). Experiments reveal that adding our manually created filters increase the prediction accuracy on a majority of the 128 datasets of the UCR Archive. They also show that hand-crafted filters and learned filters are complementary to obtain the best preforming models. This work is the first step in proposing a catalog of generic and fixed filters that could be useful in a large range of applications to improve deep models accuracy for time series classification.**

***Index Terms*—Time Series Classification, Convolution Neural Networks, Pattern Recognition, Feature Engineering, hand-crafted Filters**

I. Introduction

In recent years, Time Series Classification (TSC) has seen a rising interest in the scientific community, especially after the release of the UCR archive [1], the largest archive of univariate TSC datasets. TSC has also been used for surgical skills prediction [2], physical rehabilitation assessment [3] and remote sensing [4] In recent work, Deep Learning approaches have been used for time series analysis. These approaches include classification [5]–[7], clustering [8], [9], knowledge distillation [10], adversarial attacks [11], [12], data augmentation [13], [14], etc. In almost every cases, it is shown that Convolution Neural Networks (CNNs) can perform better than other approaches for capturing relevant features on time series. The feature extraction consists in finding linear combinations between consecutive time steps of a fixed size. The deeper the model is, the more it increases its receptive field. This represents the input space that a point in a certain depth of the network depends on. The larger the receptive field is, the more beneficial it is for the model.

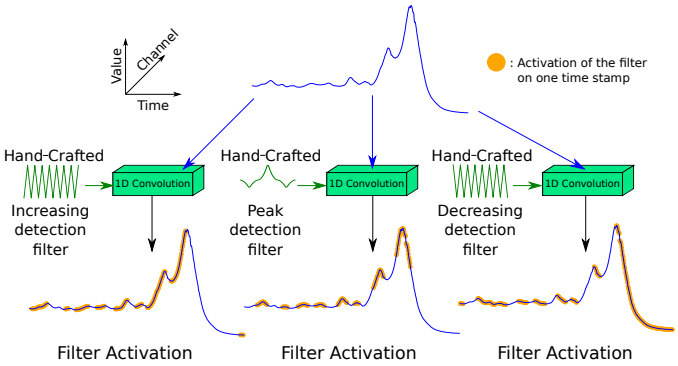

Fig. 1. Three hand-crafted filters detecting: (1) increasing trends, (2) decreasing trends and (3) peaks in a time series. The orange points indicates on which time stamps the filters is activated after being convolved with an input time series from the Meat dataset of the UCR Archive.

In Computer Vision, many work have been propose to explain the features detected by CNNs on images. It has been shown that most of the time, the first convolution layers allow to detect edges in an image. The deeper layers are then able to detect more complex features such as change in material, background and orientation. This motivated researchers to create some hand-crafted convolution filters, instead of learning them, in order to detect these features [15]–[17].

In this work, we address the TSC problem and propose some hand-crafted filters dedicated to time series which, to the best of our knowledge, has not been done before. We propose three types of filters, the increasing and decreasing trend detection filters and the peak detection filter. A summary of the hand-crafted filters proposed in this paper can be seen in Figure 1. Given that in time series domain we work in a one dimensional axis , the first types of patterns that can be useful to detect for classifying time series are the increasing and decreasing trends. Moreover, some type of datasets have a lot of oscillations (ups and downs) with a high frequency. This could perturb the classification if a model only takes into consideration the increasing and decreasing trends. That is why we also consider another type of pattern that can be useful to detect, the peaks.

Deep Learning models use back propagation in order to minimize a loss function on a given input. The problem with this last algorithm is the error propagation from layer to layer

in the neural network. In other words, if an error occurs in the last layer of the network, the error it generates is propagated to the first layer. Hence, Deep Learning model can find it hard to learn a general filter that satisfies all of the samples in the dataset with the lowest loss possible. To fix this, a trivial approach would be to add more filters to be learned. This could lead to great performances but also encourage over-fitting in some cases. An additional problem with this trivial solution is its cost of memory and time complexity. To overcome these issues, other approaches have been employed like applying regularization to the filters or using dropout to generalize better and be more robust. In this work we propose an original approach using hand-crafted filters we create, in parallel to the filters learned by the Deep Learning model. In this way, the network focuses more on types of filters that were not manually created.

To evaluate the impact of hand-crafted filters on TSC performance, we propose three models adapted from previous architectures, FCN [5] and InceptionTime [6]. We thus evaluate three hybrid models and show that with the help of the generic hand-crafted filters, the model can focus on other types of filters and not re-calculate the same filters (hand-crafted ones). We would like to point out that our hand-crafted filters proposed in this paper are independent of the dataset. They thus act as generic filters able to extract similar features for any input time series. This does not hold true for the learned filters of a Convolutional Neural Network (CNN) given that the filters vary from a dataset to another due to the learning phase.

Our main contributions in this work are:

- We propose new hand-crafted filters to detect peaks, increasing and decreasing trends.
- We create novel hybrid models of existing architectures by incorporating hand-crafted filters within them.
- We evaluate these hybrids models on the UCR Archive and show the large impact of hand-crafted filters on performances.

The rest of the paper is organized as follows: in Section II we review some state of the art methods addressing TSC, in Section III we describe in details the proposed hand-crafted filters and introduce our resulting hybrid models, in Section IV we evaluate and discuss the impact of our hand-crafted filters for TSC and finalize with a conclusion in Section V.

II. Related Work

TSC aims at associating a given time series to a corresponding class label using a classifier. This classifier can be for instance a linear regression, a decision tree, a Nearest Neighbor or a Deep Neural Network. The following paragraphs review some of the most relevant existing approaches addressing TSC.

A. On Raw Data

A first set of TSC methods worked on the definition of a metric computing the similarity between two raw time series. Hence, the Dynamic Time Warping (DTW) metric has been widely employed as it allows to compare two raw time series independently to their temporal distortions. For instance, in [18], the authors employed the DTW metric with a nearest neighbor classifier (NN-DTW). The authors of [19] further proposed an adaptation of DTW, shapeDTW, that avoids aligning point to point between two time series. The resulted metric, when used with the Nearest Neighbor algorithm (NN-ShapeDTW), significantly improved the TSC performance, setting a new benchmark on the UCR Archive. In addition, the authors of [20] introduced a way of averaging time series using the DTW metric. This algorithm aligns point to point two time series and average over the aligned time steps. This method was evaluated in a TSC problem using NN-DTW.

B. Deep Neural Networks

Deep Learning was proven to be powerful in performance on images since [21]. Several adaptations have been proposed for time series [5], [22]–[24]. A Fully Convolution Network was proposed by [5], consisting of three convolution blocks followed by a batch normalization and a ReLU activation. The authors also adapted the Residual Network (ResNet) for time series, originally proposed in [25] for images. The authors used three residual blocks, each made of a FCN with different number of filters. More recently, in [6], the authors used the idea of residual connection but instead of regular convolution blocks, they proposed multiple convolution layers in parallel in order to capture patterns at different scale using different kernel sizes. This more complex architecture, named InceptionTime, is currently the best performing Deep Learning-based approach for TSC. Differently, while more and more complex Deep Learning architectures has been built for TSC, other work focused on alleviating such increasing complexity. For instance, in [10], knowledge distillation has been employed in order to decrease the size of the FCN model without decreasing too much the TSC performances.

C. Convolutions

As explained before, the most efficient Deep Learning approaches are based on convolutions. This is emphasized in [26] where several Deep Learning approaches are compared. This review shows that the use of convolutions allows to capture relevant features on time series. Instead of learning these convolutions, the authors in [27] introduced ROCKET, a novel approach using random convolution kernels and classifying the captured information with a RIDGE classifier [28]. ROCKET beats the state of the art Deep Learning architectures for TSC. Some variations of ROCKET have been introduced in other work such as MINIROCKET [29], S-ROCKET [30], MultiRocket [31], HYDRA [32].

The success of these methods motivated us to explore the use of hand-crafted convolution filters for TSC. Such hand-crafted filters have been widely employed in image processing like the Sobel filters [15], [16] for edge detection. In particular, these Sobel filters have been successfully employed for boosting image classification in [17]. In this work, we aimed at building hand-crafted convolution filters dedicated to time

series and explore their impact on TSC when combined with convolutions trained using Deep Learning.

III. Method

A. Definitions

Before describing our proposed method, we introduce important definitions that are employed in the rest of this paper.

a) Univariate Time Series: Let $\mathbf{x}$ be a univariate time series of length L, a sequence of data points equally separated in time.

b) Univariate Time Series Dataset: A dataset $\mathcal{D} = \{(\mathbf{x}_0, y_0), ..., (\mathbf{x}_N, y_N)\}$ is a set of N pairs of univariate time series of length L and a label y associated to it.

c) One Dimensional Convolution: An operation using a filter $\mathbf{w}$ of length k on a time series $\mathbf{x}$ to obtain $\mathbf{s} = \mathbf{x} * \mathbf{w}$ as follows:

$$\forall t \in [0; L-1] \quad \mathbf{s}[t] = \sum_{i=0}^{k-1} \mathbf{x}[t+i].\mathbf{w}[i] \tag{1}$$

d) Activation of Filter: When the convolution operation results in a positive response, the filter is considered as activated. In this paper, such an activation is depicted by a orange point in the following figures.

e) Increasing Trend: A sub-sequence of a time series $\mathbf{x}$ where the values are strictly increasing in time.

f) Decreasing Trend: A sub-sequence of a time series $\mathbf{x}$ where the values are strictly decreasing in time.

g) Stationary Trend: A sub-sequence of a time series $\mathbf{x}$ where the values vary of a small difference ϵ.

h) Peak: A sub-sequence of a time series $\mathbf{x}$ where the values changed with a large variation increasingly and then decreasingly.

B. hand-crafted Filters For Time Series

In this paper, we propose hand-crafted filters adapted to time series in order to detect specific patterns. These hand-crafted filters are described in the following paragraphs.

1) Increasing Trend Detection Filter: In order for a filter to detect an increasing trend, it should detect the difference in values between time steps. Therefore, we define an increasing trend filter of length k as follows: $\mathbf{w_{I_k}} = [(-1)^{(i+1)}$ for $i \in \{0, ..., k-1\}$. To ensure that there would not be any time steps left untouched when applying the convolution, k should be an even number. For instance, the increasing trend detection filter of length 12 is defined as:

$$\mathbf{w_{I_{12}}} = [-1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1] \tag{2}$$

The resulting convolution operation with an increasing trend detection filter of size $k = 16$ can be seen in Figure 2. We can observe that activation points in orange are mainly localized in the increasing parts of the time series.

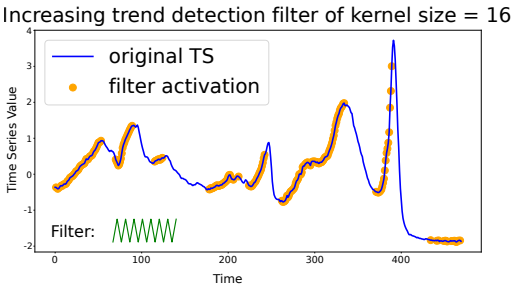

Fig. 2. hand-crafted increasing trend detection filter of length $k = 16$ applied on the Beef dataset of the UCR Archive. The time steps where the filter is activated (in orange) is only on the increasing intervals of the time series (in blue).

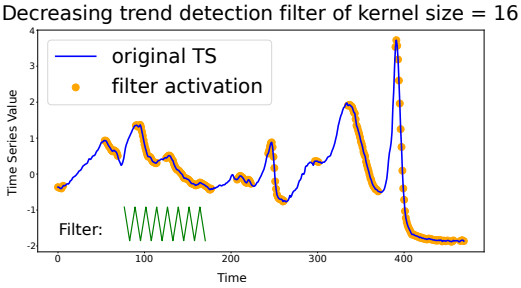

Fig. 3. hand-crafted decreasing trend detection filter of length $k = 16$ applied on the Beef dataset of the UCR Archive. The time steps where the filter is activated (in orange) is only on the decreasing intervals of the time series (in blue).

2) Decreasing Trend Detection Filter: Similarly to the increasing trend detector, the decreasing trend detection filter should detect the difference in values between time steps. However, it should be activated when applied on a decreasing interval. Hence, we define the decreasing trend detection filter of length k as follows: $\mathbf{w_{D_k}} = [(-1)^i$ for $i \in \{0, ..., k-1\}]$. For the same reason as the increasing trend detection filter, k should be an even number. For instance, the decreasing trend detection filter of length 12 is defined as:

$$\mathbf{w_{D_{12}}} = [1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1] \tag{3}$$

Figure 3 shows the application of the decreasing detection filter of size $k = 16$ on a time series. This filter is mainly applied on decreasing parts of the time series, as depicted by orange points.

3) Peak Detection Filter: We consider a peak as an increasing trend followed by a decreasing trend with a large variation between them. To capture such a peak in a time series, it requires to detect a change of convexity. To do that, we propose to mimic the shape of the negative second derivative of the Gaussian function illustrated in Figure 4. Such shape can be defined by using the squared parabolic function $f(x) = x^2$.

As our goal is to create fixed and generic filters than can be used for any datasets, we do not directly employ the Gaussian filter in order to avoid randomness of choosing the mean and variance in the filters. Hence, we propose to create this kind of filter by dividing it into three parts of equal length as follows:

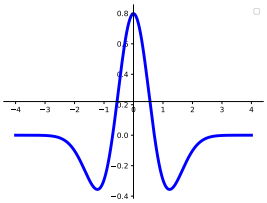

Fig. 4. Inverted second derivative Gaussian. This function is mimicked using a second order polynomial . By this approach we created the hand-crafted peak detection filter. We held off using the Gaussian filter to avoid having hyperparameters as which mean and variance to use.

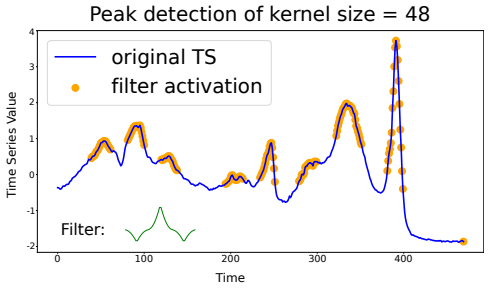

Fig. 5. hand-crafted peak detection filter of length $k = 48$ applied on the Beef dataset of the UCR Archive. The time steps where the filter is activated (in orange) is only on the peaks intervals of the time series (in blue).

(1) a negative parabolic part detecting the first increasing trend, (2) a positive parabolic part detecting the peak , (3) a negative parabolic part detecting the decreasing trend. For instance, the peak detection filter of length 12 is defined as:

$$\mathbf{w_{P_{12}}} = [-0.25, -1, -1, -0.25, 0.5, 2, \\ 2, 0.5, -0.25, -1, -1, -0.25] \quad (4)$$

A visualization of the convolution operation using the peak detection filter of size $k = 48$ can be seen in Figure 5.

In order to capture variable length patterns we use in this paper a set of n variations of the hand-crafted increasing, decreasing and peak detection filters. Once the hand-crafted filters are defined, our aim is to employ them as generic features extractors within Deep Learning architectures for TSC. In this work, we consider two different architectures: the Fully Convolutional Network (FCN) and the InceptionTime. These architectures and their required adaptation to incorporate hand-crafted filters are described in the two following subsections.

C. The Fully Convolution Network (FCN) Adaptations

1) Two Layers FCN: We first consider the FCN architecture [5], [26] because of it's simplicity compared to other architectures [5], [6], [26], [27]. In order to use the hand-crafted filters defined above, we create an adaptation of FCN called **Customs Only FCN** (CO-FCN). A detailed example of the CO-FCN architecture is depicted in Figure 6. In this first adaptation, we simply replaced the first layer of FCN by n variations of each hand-crafted filter. The rest of the architecture in FCN is not changed according to the original architecture proposed in [5].

2) Hybrid FCN: The CO-FCN architecture assumes that learning convolutions at the first layer is not required. However, although the hand-crafted filters allow to detect three different kind of patterns, we believe that the model can find additional relevant patterns. Hence, we design the **Hybrid FCN** (H-FCN), whose an example can be seen in Figure 7. Instead of replacing the entire first convolution layer by the hand-crafted filters, we propose to enhance it. In order to do that, features extracted using our hand-crafted filters are concatenated to the features extracted by the first trainable convolution layer of the original FCN. The original FCN proposed in [5] is made of three convolution blocks with 128,256 and 128 filters respectively. In this work, we believe that with the help of our hand-crafted filters, it is sufficient to limit the number of trainable filters to 64,128 and 64, respectively. We note that as our hand-crafted convolution filters do not include a bias, we also remove the bias from the trainable convolution layers.

D. The InceptionTime Adaptation

1) Hybrid Inception: We now consider the more complex architecture Inception [6] as it is currently the best performing Deep Learning model for TSC on the UCR Archive. In order to incorporate our hand-crafted filters into Inception, we propose the **Hybrid Inception** (H-Inception). Similarly to H-FCN, we concatenate the features captured using our hand-crafted filters with those captured by the first Inception block. The rest of the architecture remains the same as the original. We note that unlike H-FCN, we do not reduce the number of trainable filters per convolution layer as it is already low in the Inception model.

2) Hybrid InceptionTime: In [6], an ensemble of five Inception models is also proposed. We adapt the same idea with the **H-InceptionTime** which is an ensemble of five different H-Inception models. A detailed architecture of a H-Inception model can be seen in Figure 8.

IV. Experimental Evaluation

A. Datasets And Implementation Details

For evaluating the proposed models, we use the UCR Archive 2018 [1], made of 128 datasets of labeled univariate time series. For a fair comparison with existing approaches, z-normalization is applied on each dataset. We trained each model using the Adam optimizer [33] with a learning rate decay monitoring the training loss. The best model obtaining the best loss during training is kept for evaluation on the test set. In order to be less dependant on random initialization, the whole process is repeated five times. Hence, the results shown in the rest of the paper are averaged over five different initialization. All of the experiments were done on a NVIDIA GeForce GTX 1080 with 8GB of memory. The code is available here : https://github.com/MSD-IRIMAS/CF-4-TSC.

B. Results On Adapted Architectures

We compared the performance of our adapted architectures using hand-crafted filters with the original models. For a

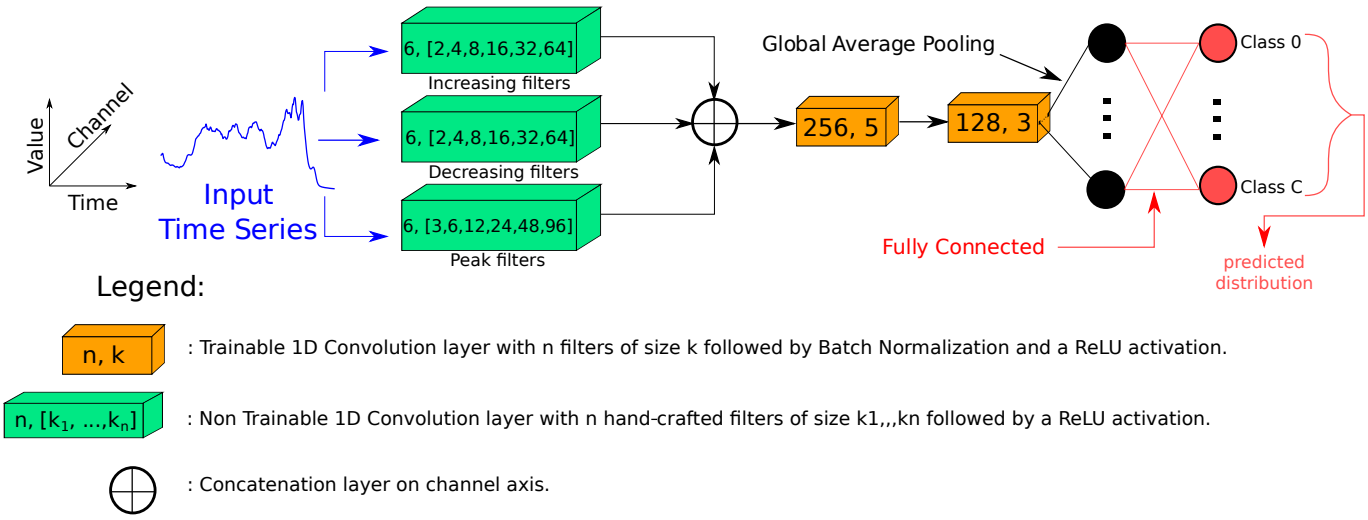

Fig. 6. An example of the Two Layers FCN architecture. The input time series is fed to the non learned hand-crafted filters in order to extract the corresponding features. Features are then concatenated and fed to the rest of the architecture made of two convolution blocks (in orange) each followed by a Batch Normalization and a ReLU activation. A global average pooling is then applied on the second convolution block (black connections) followed by a fully connected classification layer (red connections).

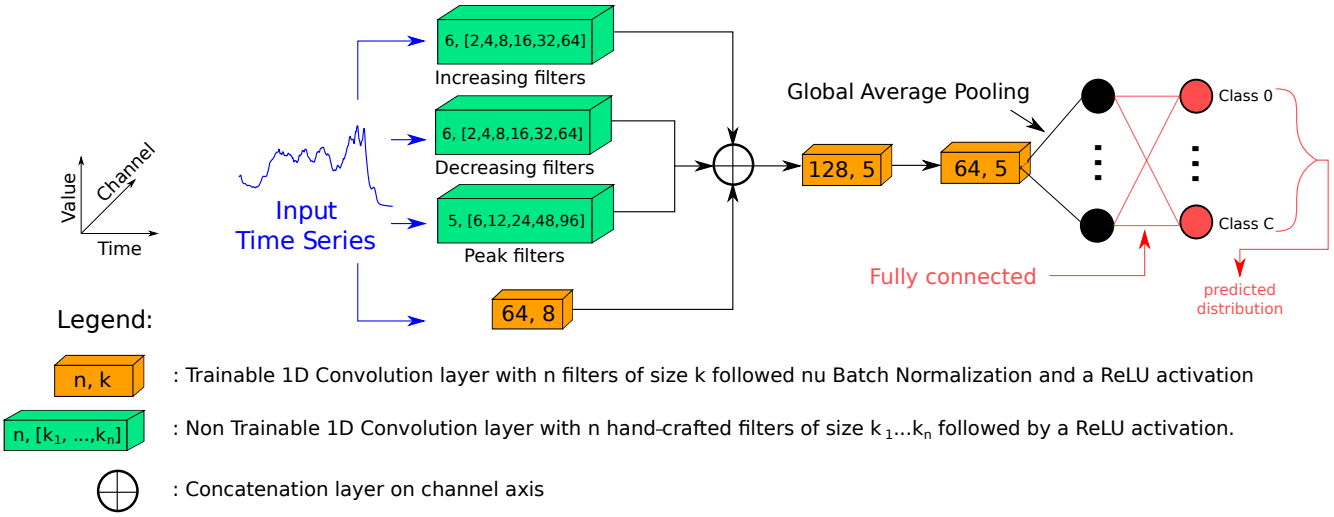

Fig. 7. An example of the Hybrid FCN architecture. The input time series is fed to a convolution block (in orange, bottom left) and to the hand-crafted filters (in green). Features are then concatenated and fed to the rest of the architecture including two convolution blocks in series (in orange) each followed by a Batch Normalization and a ReLU activation. A global average pooling is then applied on the third convolution block (black connections) followed by a fully connected classification layer (red connections).

pair of models, we compared the obtained accuracy on each dataset and computed the number of wins, ties and losses. Such comparative results are reported using Win/Tie/Loss one-vs-one plot, as shown in Figures 9, 10 and 11. These plots show the Win/Tie/Loss count between two different classifiers on the 128 datasets of the UCR Archive. Each point in the plots of represents a single dataset of the UCR Archive. The axes show the accuracy of each classifier (averaged over five initialization) between 0 and 1. Moreover, in order to assess how significant the comparison is, the Wilcoxon Signed Rank Test [34]–[36] is performed for each pair of classifier. The resulting statistical measure, the P-value, is shown in the legend of each plot. If the P-Value between two classifiers is below a threshold, it means they are significantly statistically different in performance. Usually the threshold of this P-Value for this kind of decision is 0.05.

1) CO-FCN: To train this model we used the same formula of batch size used in the original FCN and the same number of epochs [5], [26]. Hence, the batch size is the minimum between 16 and the number of samples in the training set divided by 10, while the number of epochs is set to 2000. For the increasing and decreasing detection filters we used 6 variations of lengths $[2^i$ for $i \in \{1, ..., 6\}]$. For the peak detection filters we used 6 variations of lengths $[3, 6, 12, 24, 48, 96]$.

In Figure 9, the Win/Tie/Loss plot shows that the CO-FCN obtains better performances than original FCN on the majority

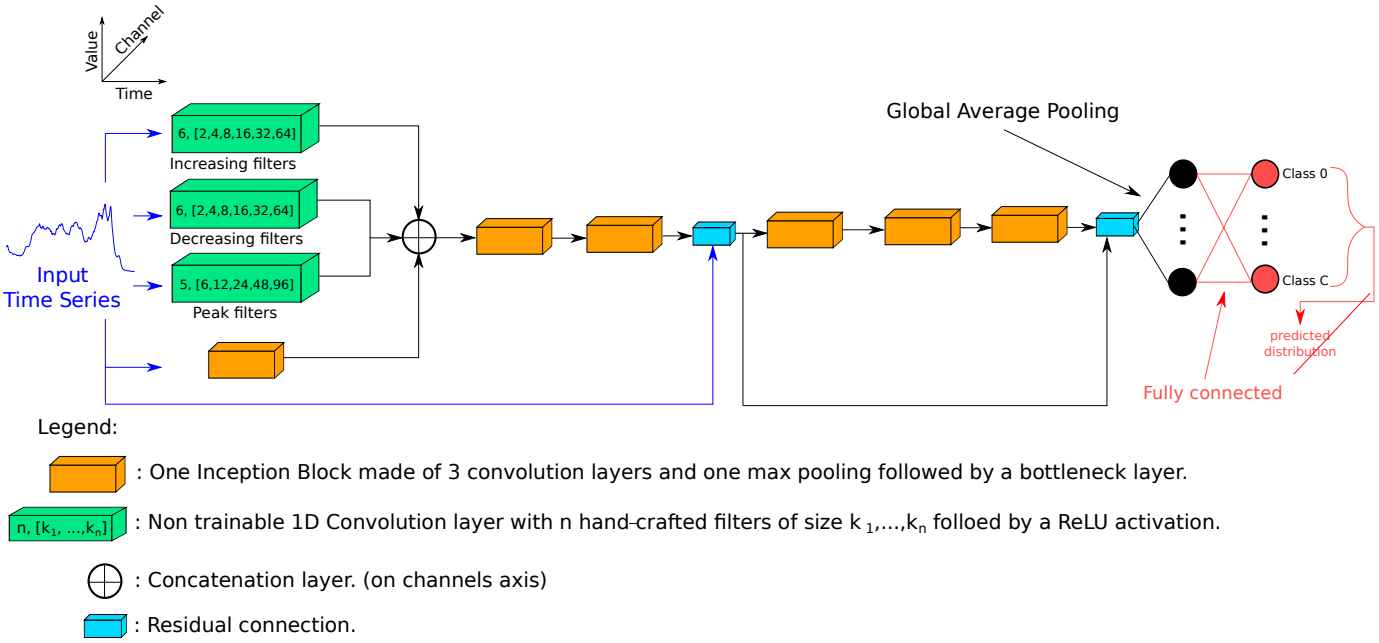

Fig. 8. An example of the Hybrid Inception architecture. The input time series is fed to the non learned hand-crafted filters and to an Inception block. Features are then concatenated and fed to the rest of the architecture consisting of 5 Inception blocks with residual connection as in the original model. A global average pooling is then applied on the second residual block (black connections) followed by a fully connected classification layer (red connections).

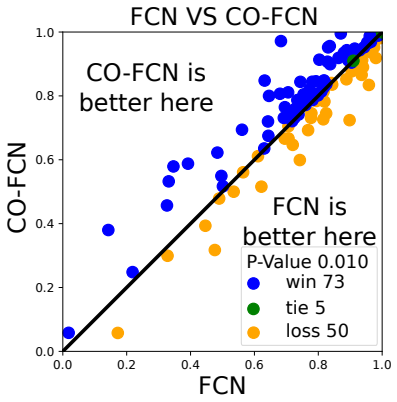

Fig. 9. One-VS-One plot showing the Win/Tie/Loss count and the P-Value when comparing the proposed CO-FCN and the original FCN on the 128 datasets of the UCR Archive.

of datasets. The low P-Value suggests that this difference is significantly meaningful. This shows that in most of the cases hand-crafted filters can be used to extract features at the first in replacement of learn convolutions. However, it can also be seen that for 50 datasets, the original FCN works better than our CO-FCN. This suggests that combining hand-crafted filters with trainable filters may improve the overall results. This combination is done in our H-FCN model.

2) H-FCN: Similarly to the previous section, for training our H-FCN model, we used the same batch size and number of epochs as the original FCN so that the comparison can be as fair as possible. For the increasing and decreasing detection filters we employed 6 variations of lengths $[2^i$ for $i \in \{1, ..., 6\}]$. For the peak detection filters we used 5 variations of lengths $[6, 12, 24, 48, 96]$.

In Figure 10, we can see that the H-FCN model obtains significantly better performances than the original FCN model (top left) but also than ResNet (top right). However, when comparing H-FCN to the more complex Inception model (bottom left), we can see that H-FCN obtains slightly lower results. In addition we compared our H-FCN model to the state-of-the-art InceptionTime approach. For a fair comparison, we also created an ensemble of H-FCN models resulting in H-FCNTime. Results reported in Figure 10 (bottom right) show that InceptionTime is significantly better than H-FCNTime. This suggests that even if hand-crafted filters allow to improve the performance of FCN, the limitations of the FCN architecture restrain the capture of more complex features to be competitive with deeper architectures.

3) H-InceptionTime: In order to evaluate the contribution of our hand-crafted filters in a deeper architecture, we tackled the InceptionTime model. As in the original work [6], we employed a batch size of 64 and trained the model for 1500 epochs using an Adam optimizer with a learning rate decay. For the increasing and decreasing detection filters we used 6 variations of lengths $[2^i$ for $i \in \{1, ..., 6\}]$. For the peak detection filters we considered 5 variations of lengths $[6, 12, 24, 48, 96]$. We compared our adapted H-Inception and H-InceptionTime models with the state of the art models Inception, InceptionTime and ROCKET. Results, averaged over five runs, are reported in Figure 11 using Win/Tie/Loss plots.

We can first observe that H-Inception has more wins than Inception (top left) with a P-Value less than 0.05, meaning that both models are significantly different. Moreover, the

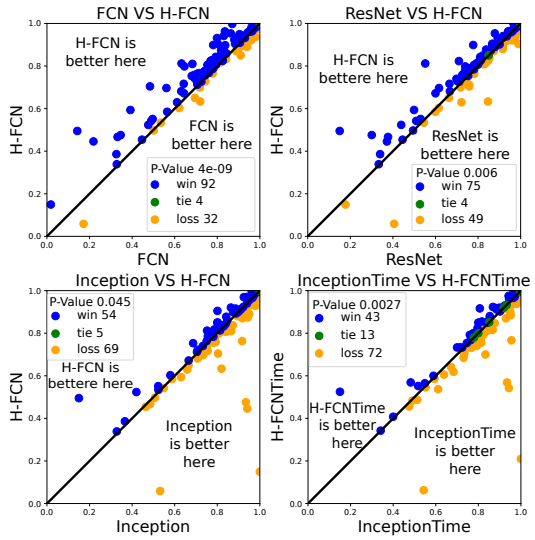

Fig. 10. One-VS-One plot showing the Win/Tie/Loss count and the P-Value when comparing the proposed H-FCN with the original FCN, ResNet, Inception and InceptionTime models on the 128 datasets of the UCR Archive.

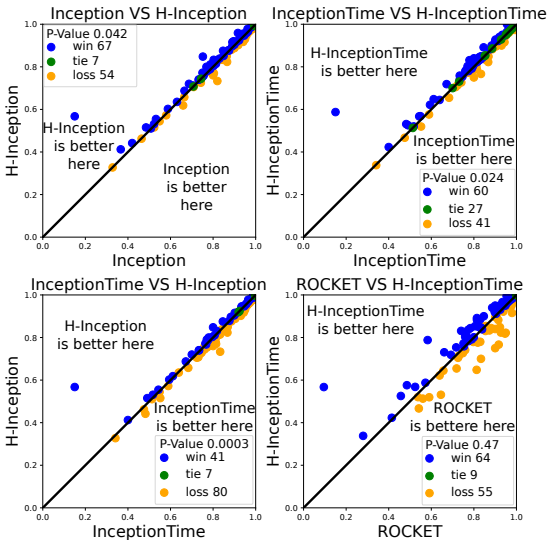

Fig. 11. One-VS-One plot showing the Win/Tie/Loss count and the P-Value when comparing the proposed H-Inception, H-InceptionTime with the original Inception, InceptionTime and ROCKET models on the 128 datasets of the UCR Archive.

comparison between H-Inception and InceptionTime (bottom left) shows that InceptionTime, thanks to the ensemble, is significantly better than our H-Inception model. Nevertheless, after comparing the performances of the ensemble methods H-InceptionTime and InceptionTime (top right), we can see that our H-InceptionTime model significantly overcomes the original Inception time. Finally, by comparing H-InceptionTime with ROCKET (bottom right), we can notice that H-InceptionTime beats ROCKET in the Win/Tie/Loss count over the 128 datasets of the UCR Archive. However, the P-Value suggests that their is not a significant difference between these two classifiers.

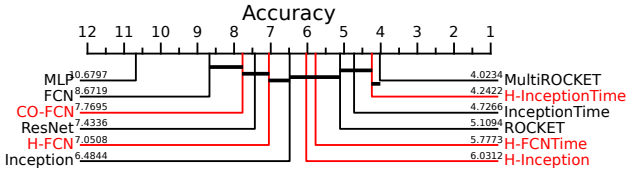

Fig. 12. CD-Diagram showing the average rank between multiple classifiers on the 128 datasets of the UCR Archive. We colored our approaches that include the hand-crafted filters in red.

C. Multi-Classifiers Comparison

After comparing pairs of classifiers, we also proposed to compare our adapted models using hand-crafted filters with existing approaches all together. To do that, we used the Critical Difference Diagram (CD-Diagram) proposed in [36]. A comparison using this method is reported in Figure 12. The CD-Diagram shows the ranking of each method averaged over the 128 datasets of the UCR Archive. In addition, the P-Value is calculated between each pair of classifiers. The algorithm decides if two classifiers are significantly different by comparing the P-Value with a threshold followed by the Holm Correction [37], [38]. A black line is drawn between two classifiers if they are not significantly different. It can be seen in the CD-Diagram of Figure 12 that H-InceptionTime is the second best classifier on the average rank metric. The best classifier is MultiROCKET [31]. Moreover, by analyzing the ranking of our models highlighted in red with their original counterparts, we can clearly see that for each architecture, the addition of the hand-crafted filters allows to increase the performance.

In addition to the CD-Diagram comparison, and motivated by [31], we also compared all these methods using the pairwise statistical significance matrix shown in Figure 16. Similarly to [31], each cell in the matrix shows the Win/Tie/Loss count between two classifiers and the P-Value calculated using the Wilcoxon signed-rank test. In addition, we also computed the difference in accuracy over the 128 datasets of the UCR Archive for both winning and loosing cases. For instance, the cell comparing H-InceptionTime with InceptionTime shows that H-InceptionTime obtains on average 2.31% higher accuracy on the 60 winning cases. Conversely, it obtains on average 1.35% lower accuracy on the 41 loosing cases. To emphasize such a difference in accuracy, a colormap is used in Figure 16. Higher differences in winning cases are shown with warm colors, while loosing cases are depicted with cold colors. The figure shows that our H-InceptionTime model is significantly more accurate than most of the methods.

D. Trainable VS Non Trainable Filters

In order to assess the relevance of our hand-crafted filters, we proposed to compare them with the learned filters in the original models. Our first aim was to analyse if original models are learning filters similar to our hand-crafted ones. In this experiment, we considered the 128 filters learned by first layer of the original FCN model on the CinCECGTorso dataset. For

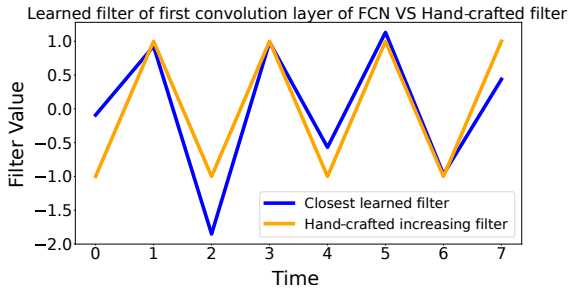

Fig. 13. hand-crafted increasing trend detection filter of size $k = 8$ and its closest learned filter on the CinCECGTorso dataset. The learned filter is from the first layer of the original FCN.

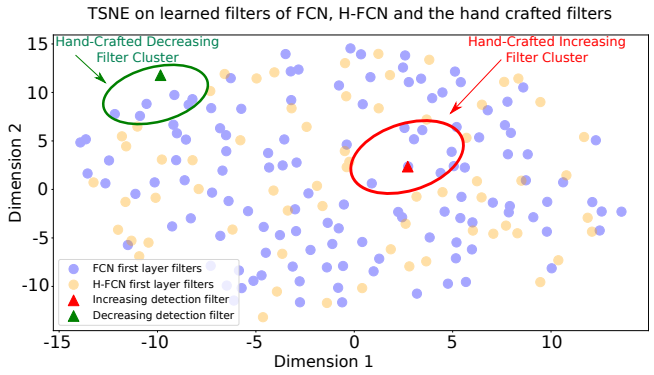

Fig. 14. T-SNE two-dimensional projection of the 128 filters learned by the first layer of the original FCN and the 64 filters learned by the the first layer of the H-FCN on the CricketY dataset of the UCR Archive. The two hand-crafted increasing and decreasing trend detection filters are also projected in the two-dimensional space. We used the DTW as a metric for the T-SNE.

each hand-crafted filter, we found the closest learned filter by computing the DTW distance for each pair of filters, after Z-normalizing the learned filters. We acknowledge that using Euclidean Distance (ED) can be a more trivial approach for comparing filters. However, we were interested in detecting similarities between filters taking into consideration the shifting between them. For this reason, DTW was more suitable than ED given its ability to consider temporal alignment.

For instance, the hand-crafted increasing filter of size $k = 8$ and its closest learned filter on CinCECGTorso dataset are shown in Figure 13. As we can see, the original FCN model learned a similar filter detecting increasing trends in the time series. The learned filter is a weighted version of our hand-crafted filter. In addition to that, the shape of the first part of the learned filter seems to be shaped as the peak detection filter and the rest of the filter as an increasing trend detection. This suggests that FCN learned how to construct a filter that captures multiple patterns at the same time. This is in line with our motivation of not only using hand-crafted filters in the first layer but also allowing some filters to be learned, as done in our H-FCN model.

In addition, we also proposed to asses the impact of hand-crafted filters incorporated in the H-FCN model. We compared the 64 learned filters by the first layer of H-FCN and the 128 filters learned by the first layer of the original FCN. We also incorporated our hand-crafted increasing and decreasing filters in the comparison. Once the DTW distance computed for each pair of filters, we project them into into a two-dimensional space using the T-distributed Stochastic Neighbor Embedding (T-SNE) [39], as shown in Figure 14 on the CricketY dataset. We can observe that H-FCN learned filters (in orange) are quite similar to the FCN learned filters (in blue). However we can notice some empty areas in the H-FCN filters distribution, as highlighted by red and green ellipsoids. This areas correspond to the hand-crafted increasing filter (red triangle) and decreasing filter (green triangle). These hand-crafted filters can then be seen as representative prototypes of several FCN learned filters. These hand-crafted filters allow to reduce the number of learned filters in H-FCN while letting the model focusing more on learning other meaningful filters.

E. Generalization With The Help Of hand-crafted Filters

Experimental results in Section IV-B demonstrated that incorporating hand-crafted filters in Deep Learning models increases the performances. However, we have shown in Section IV-D that hand-crafted filters in H-FCN are similar to some learned filters in the original FCN. These observations raise the following question: If the FCN model can learn by itself the hand-crafted filters, why does the the incorporation of hand-crafted filters in H-FCN help to be more accurate? We believe that this can be explained by the better capability of generic hand-crafted filter to generalize to unseen time series. Indeed, without any hand-crafted filters like in FCN, the model optimizes the filter weights in order to capture weighted patterns that almost perfectly fit the training set. It is then more likely to overfit on the training set, a well known issue in Deep Learning-based approaches for TSC. Conversely, in H-FCN, the incorporation of generic hand-crafted filters allows to capture general patterns that are more likely to be also relevant for unseen time series during the test phase.

In order to validate this hypothesis, we proposed to compare the training and validation curves for both FCN and H-FCN models on the FiftyWords dataset, as shown in Figure 15. We note that validation loss was computed on the test set as the UCR Archive only provides training and test sets. This was done only for monitoring the generalization behavior and not to fine-tune any hyper-parameters. We can clearly observe that the validation loss of H-FCN converges to a much lower value than the validation loss of FCN. This shows that H-FCN generalizes better on the FiftyWords dataset and explains why it obtains a 15% higher accuracy on the test set in comparison to FCN.

V. Conclusion

In this paper we addressed the problem of Time Series Classification using Convolutional Neural Network architectures. We proposed to enhance such architectures by combining the learned filters with hand-crafted filters. In particular, we designed three generic hand-crafted filters for detecting

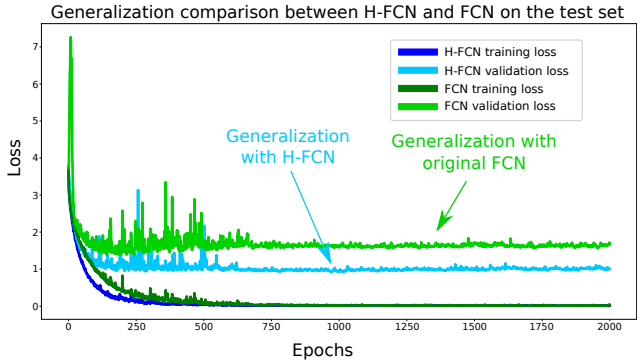

Fig. 15. Training phase of the FCN and H-FCN architectures on the FiftyWords dataset while monitoring the validation loss on the test set. The training loss of H-FCN (in blue) converges faster than the training loss of FCN (in green). Also, the validation loss of H-FCN (in cyan) is always below the validation loss of FCN (in light green). This shows that H-FCN generalizes better than FCN on this dataset.

increasing trends, decreasing trends and peaks. We evaluated the impact of incorporating these hand-crafted filters within state of the art architectures using the UCR Archive. Results demonstrated significant improvements for not only for the basic FCN architecture but also for the more complex InceptionTime. Further analysis of filters suggested that our hand-crafted filters allow better generalization on unseen datasets. We believe that this work is a first step in discovering generic convolution filters allowing to not only boost the performances of deep architectures but also reduce their number of parameters to train. As future work, we aim at investigating new hand-crafted filters by exploring other generic patterns or by combining more than one pattern to recognize.

ACKNOWLEDGEMENT

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] H. A. Dau, A. Bagnall, K. Kamgar, C.-C. M. Yeh, Y. Zhu, S. Gharghabi, C. A. Ratanamahatana, and E. Keogh, "The ucr time series archive," *IEEE/CAA Journal of Automatica Sinica*, vol. 6, no. 6, pp. 1293–1305, 2019.

[2] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Evaluating surgical skills from kinematic data using convolutional neural networks," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2018, pp. 214–221.

[3] M. Devanne, O. Rémy-Néris, B. Le Gals-Garnett, G. Kermarrec, A. Thépaut *et al.*, "A co-design approach for a rehabilitation robot coach for physical rehabilitation based on the error classification of motion errors," in *2018 Second IEEE International Conference on Robotic Computing (IRC)*. IEEE, 2018, pp. 352–357.

[4] M. Rußwurm, C. Pelletier, M. Zollner, S. Lefèvre, and M. Körner, "Breizhcrops: A time series dataset for crop type mapping," *ISPRS-International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 43, pp. 1545–1551, 2020.

[5] Z. Wang, W. Yan, and T. Oates, "Time series classification from scratch with deep neural networks: A strong baseline," in *2017 International joint conference on neural networks (IJCNN)*. IEEE, 2017, pp. 1578–1585.

[6] H. Ismail Fawaz, B. Lucas, G. Forestier, C. Pelletier, D. F. Schmidt, J. Weber, G. I. Webb, L. Idoumghar, P.-A. Muller, and F. Petitjean, "Inceptiontime: Finding alexnet for time series classification," *Data Mining and Knowledge Discovery*, vol. 34, no. 6, pp. 1936–1962, 2020.

[7] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Transfer learning for time series classification," in *2018 IEEE international conference on big data (Big Data)*. IEEE, 2018, pp. 1367–1376.

[8] B. Lafabregue, J. Weber, P. Gançarski, and G. Forestier, "End-to-end deep representation learning for time series clustering: a comparative study," *Data Mining and Knowledge Discovery*, vol. 36, no. 1, pp. 29–81, 2022.

[9] T. Terefe, M. Devanne, J. Weber, D. Hailemariam, and G. Forestier, "Time series averaging using multi-tasking autoencoder," in *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, 2020, pp. 1065–1072.

[10] E. Ay, M. Devanne, J. Weber, and G. Forestier, "A study of knowledge distillation in fully convolutional network for time series classification," in *International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2022, pp. 1–8. [Online]. Available: https://doi.org/10.1109/IJCNN55064.2022.9892915

[11] G. Pialla, H. I. Fawaz, M. Devanne, J. Weber, L. Idoumghar, P.-A. Muller, C. Bergmeir, D. Schmidt, G. Webb, and G. Forestier, "Smooth perturbations for time series adversarial attacks," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2022, pp. 485–496.

[12] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Adversarial attacks on deep neural networks for time series classification," in *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2019, pp. 1–8.

[13] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Data augmentation using synthetic data for time series classification with deep residual networks," *ArXiv e-prints*, pp. arXiv–1808, 2018.

[14] G. Pialla, M. Devanne, J. Weber, L. Idoumghar, and G. Forestier, "Data augmentation for time series classification with deep learning models," in *Advanced Analytics and Learning on Temporal Data (AALTD)*, 2022.

[15] V. Bogdan, C. Bonchiș, and C. Orhei, "Custom extended sobel filters," *arXiv preprint arXiv:1910.00138*, 2019.

[16] W. Gao, X. Zhang, L. Yang, and H. Liu, "An improved sobel edge detection," in *2010 3rd International conference on computer science and information technology*, vol. 5. IEEE, 2010, pp. 67–71.

[17] N.-D. Hoang and Q.-L. Nguyen, "Fast local laplacian-based steerable and sobel filters integrated with adaptive boosting classification tree for automatic recognition of asphalt pavement cracks," *Advances in Civil Engineering*, vol. 2018, 2018.

[18] A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh, "The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances," *Data mining and knowledge discovery*, vol. 31, no. 3, pp. 606–660, 2017.

[19] J. Zhao and L. Itti, "shapedtw: Shape dynamic time warping," *Pattern Recognition*, vol. 74, pp. 171–184, 2018.

[20] F. Petitjean, A. Ketterlin, and P. Gançarski, "A global averaging method for dynamic time warping, with applications to clustering," *Pattern recognition*, vol. 44, no. 3, pp. 678–693, 2011.

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[22] Y. Zheng, Q. Liu, E. Chen, Y. Ge, and J. L. Zhao, "Exploiting multi-channels deep convolutional neural networks for multivariate time series classification," *Frontiers of Computer Science*, vol. 10, no. 1, pp. 96–112, 2016.

[23] Z. Cui, W. Chen, and Y. Chen, "Multi-scale convolutional neural networks for time series classification," *arXiv preprint arXiv:1603.06995*, 2016.

[24] A. Le Guennec, S. Malinowski, and R. Tavenard, "Data augmentation for time series classification using convolutional neural networks,"

in *ECML/PKDD workshop on advanced analytics and learning on temporal data*, 2016.
[25] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
[26] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Deep learning for time series classification: a review," *Data mining and knowledge discovery*, vol. 33, no. 4, pp. 917–963, 2019.
[27] A. Dempster, F. Petitjean, and G. I. Webb, "Rocket: exceptionally fast and accurate time series classification using random convolutional kernels," *Data Mining and Knowledge Discovery*, vol. 34, no. 5, pp. 1454–1495, 2020.
[28] C. Peng and Q. Cheng, "Discriminative ridge machine: A classifier for high-dimensional data or imbalanced data," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 6, pp. 2595–2609, 2020.
[29] A. Dempster, D. F. Schmidt, and G. I. Webb, "Minirocket: A very fast (almost) deterministic transform for time series classification," in *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*, 2021, pp. 248–257.
[30] H. Salehinejad, Y. Wang, Y. Yu, T. Jin, and S. Valaee, "S-rocket: Selective random convolution kernels for time series classification," *arXiv preprint arXiv:2203.03445*, 2022.
[32] A. Dempster, D. F. Schmidt, and G. I. Webb, "Hydra: Competing
[31] C. W. Tan, A. Dempster, C. Bergmeir, and G. I. Webb, "Multirocket: Multiple pooling operators and transformations for fast and effective time series classification," *Data Mining and Knowledge Discovery*, pp. 1–24, 2022.
convolutional kernels for fast and accurate time series classification," *arXiv preprint arXiv:2203.13652*, 2022.
[33] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
[34] F. Wilcoxon, "Individual comparisons by ranking methods," in *Breakthroughs in statistics*. Springer, 1992, pp. 196–202.
[35] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *The Journal of Machine Learning Research*, vol. 7, pp. 1–30, 2006.
[36] A. Benavoli, G. Corani, and F. Mangili, "Should we really use post-hoc tests based on mean-ranks?" *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 152–161, 2016.
[37] S. Holm, "A simple sequentially rejective multiple test procedure," *Scandinavian journal of statistics*, pp. 65–70, 1979.
[38] S. Garcia and F. Herrera, "An extension on" statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons." *Journal of machine learning research*, vol. 9, no. 12, 2008.
[39] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne." *Journal of machine learning research*, vol. 9, no. 11, 2008.

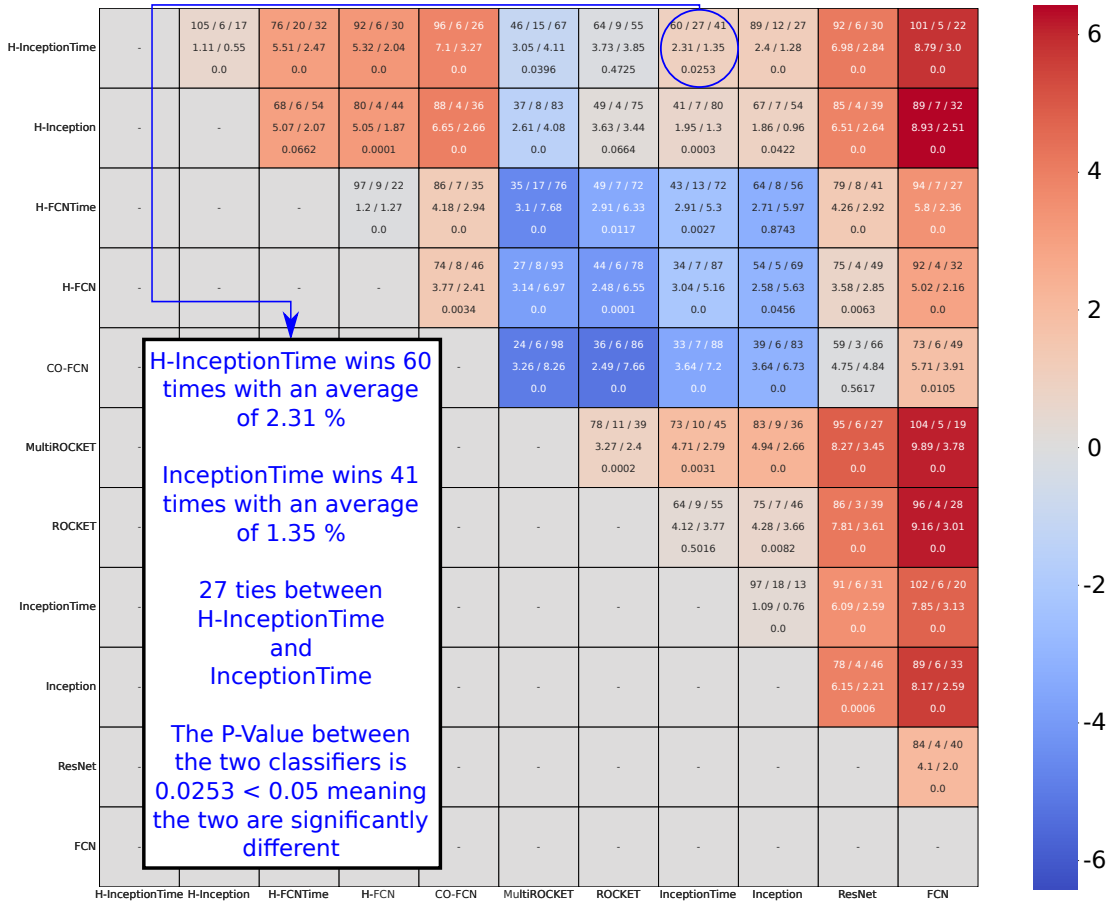

Fig. 16. Statistical significance matrix showing the pairwise comparison between classifiers over the 128 datasets of the UCR Archive. Our proposed models and existing architectures are included in this comparison. Each cell represents three information for comparing two classifiers. The first line shows the Win/Tie/Loss count, the wins are for the method on the horizontal line. The second line shows the average in difference of accuracy on the wins / on the losses. The third line shows the P-Value between the two classifiers after applying the Wilcoxon signed-rank test. A heatmap is applied using the difference of accuracy between two methods. Warm colors correspond to positive differences while cold colors correspond to negative differences.