PGA: Plug-in Inter-layer Graph Alignment for Stable Neighborhoods in Deep Face Recognition

Jinkang ZHu

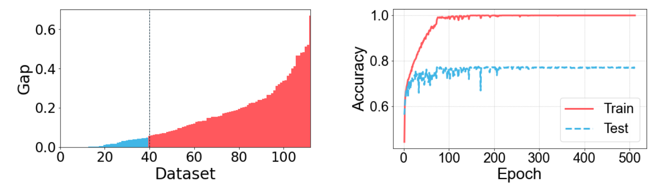
***Abstract*—Deep face recognition models often suffer from noisy local geometry (e.g., multi-cluster intra-class structure, unstable hard negatives), which degrades generalization even under strong classification losses such as ArcFace. To stabilize the learned neighborhood structure, we propose a plug-in inter-layer graph alignment (PGA) regularizer that operates within a single network. First, we build a top-k diffusion kernel at multiple intermediate stages and perform K-alignment: the kernel of a lower layer is encouraged to match the EMA-smoothed kernel of the subsequent layer on masked, reliable edges, suppressing transient/erroneous links while reinforcing persistent same-class neighbors. Second, we add a light Z-alignment on projected features to promote directional consistency across layers during early training. PGA is loss-agnostic and complements ArcFace; it introduces negligible training overhead O(Bk) and zero inference cost. [[[[introduce experiments ...]]]] and improves neighbor stability and sub-cluster connectivity, with robustness to different batch sampling schemes (e.g., random vs. PK).**

***Index Terms*—Face Recognition, Inter-Layer Graph Alignment, Self-Distillation, Diffusion Kernel, kNN Graph, EMA Target, ArcFace, Neighborhood Consistency, Regularization.**

I. Introduction

Time-series Classification (TSC) is one of the most challenging tasks in data mining [1]. In recent years, with the remarkable success of deep neural networks (DNNs) in Computer Vision (CV) [2], [3], [4], many researchers have tried to employ DNNs in TSC due to the similarity between time-series data (One-dimensional sequence) and image data (Two-dimensional sequence). However, with extensive experiments on various existing DNN-based TSC approaches, we found that DNNs are easy to be overfitting on datasets from the UCR archive [5]. To be specific, several experiments are conducted for 3 typical DNNs, i.e. Fully Convolutional Networks (FCN) [6], Residual Networks (ResNet) [3], [6], and InceptionTime [7], on the UCR datasets. Selecting InceptionTime as an example, only 40 datasets have the training and test accuracy gap less than 0.05, which means the gap on the other 73 datasets is more than 0.05, where many of them are more than 0.2, as shown in Fig. 1(a). In addition, Fig. 1(b) gives the training and test accuracy with respect to epochs on a specific dataset among UCR datasets, i.e. $Crop$ dataset, where the gap becomes 0.22 around epoch 100 and keeps stationary till the end.

After thoroughly analyzing the UCR datasets, we claim that the difference between datasets in CV and those in UCR archive contributes the most to the overfitting phenomenon. In detail, we can view this from the perspective of N-shot learning, where N represents the training examples per class. In CV, datasets always contain enough training samples for

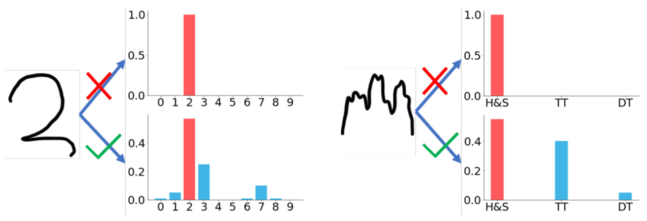


(a) The gap between training and test accuracy on 113 UCR datasets (b) The training and test accuracy w.r.t. epochs on $Crop$ dataset

Fig. 1. Two examples showing that InceptionTime, including other DNNs, is easy to be overfitting on UCR datasets

DNNs, e.g. MNIST [8] and CIFAR10 [9] are both 5000-shot learning datasets with 50000 training samples in total, and ImageNet [10] is averagely a 700-shot learning dataset with more than 14 million training samples. However, in UCR archive, 95 datasets are less than 100-shot and 120 datasets with less than 1000 training samples, while only 4 ones with more than 2000 training samples. For example, *Fungi* is a 1-shot learning dataset, and *DiatomSizeReduction* contains only 16 training samples. We conclude this as the few-shot problem in TSC. In summary, solving the few-shot problem is the key to improving accuracy. Interestingly, many state-of-the-art approaches adopted methods for alleviating overfitting without pointing out the overfitting problem, such as the Hierarchical Vote system for Collective Of Transformation-based Ensembles (HIVE-COTE) [11] with an ensemble of 37 classifiers, InceptionTime [7] with an ensemble of 5 models, RandOm Convolutional KErnel Transform (ROCKET) [12] with $L2$ regularization and Cross-Validation, etc. Among the aforementioned state-of-the-art methods, ROCKET has the best accuracy and inference time balance in practice.

In this paper, instead of employing the ordinary approaches for alleviating overfitting, e.g. $L1$ and $L2$ regularization, Batch Normalization (BN) [13], Dropout [14], early stopping, etc, we first proposed Label Smoothing based on InceptionTime [15] (LSTime) for improving the generalization ability of InceptionTime, as soft labels are closer to real-life compared to hard labels. For instance, as shown in Fig. 2(a), the true label of that handwritten number is 2. Yet, it also looks like 3 intuitively. Thus, giving a hard label 2 to that number may cause information losses. Also, in stocks, there are many meaningful chart patterns, among which Head&Shoulders (H&S), Triple Top (TT), and Double Top (DT) [16] are similar. In Fig. 2(b), the H&S chart pattern is close to TT. Therefore, we wish to keep its information of TT. As a consequence, soft labels maintain more information compared to hard labels. However, we would



(a) Soft labels for CV (b) Soft labels for TSC

Fig. 2. Two examples demonstrating the benefits of soft labels

like to get the soft labels automatically rather than to set that manually. Thus, secondly, Knowledge Distillation based on InceptionTime [17] (KDTime) is leveraged to generate the soft labels by a teacher model, which is a pre-trained network with a deep and complex architecture. The predicted labels from the teacher model represent the knowledge learned by it. After that, the knowledge (Predicted soft labels) can be distilled to help the training of a student model, which owns a relatively smaller and simpler architecture. As a consequence, Knowledge Distillation can also reduce the inference time because of the student model. At last, we claim that the teacher is not 100% correct, which means it may produce wrong soft labels and misguide the student. As the ground-truth labels have already been obtained, we propose to simply calibrate the wrong predicted soft labels, in order to maximize the accuracy of InceptionTime, called Knowledge Distillation with Calibration based on InceptionTime (KDCTime). In addition, KDCTime includes two optional calibrating strategies, i.e. KDC by Translating (KDCT) and KDC by Reordering (KDCR).

The InceptionTime model employed in LSTime, KDTime, and KDCTime is a single model instead of 5 ones, since the training and inference time are essential, which indicates the feasibility of that model. Thus, compared to an ensemble of 5 models with 6 Inception modules each in its original version, the InceptionTime model in this paper is only one with 3 Inception modules. In summary, the main contributions in this paper can be concluded as follows:

- Discovered the TSC approaches based on DNNs are normally overfitting on the UCR datasets, which is caused by the few-shot problem of those datasets. Thus, the best direction to improve the performance of DNNs is alleviating overfitting.
- Combined Label Smoothing and Knowledge Distillation with InceptionTime respectively, denoted LSTime and KDTime. LSTime trains the InceptionTime model with manually controlled soft labels, while KDTime is able to generate soft labels automatically by the teacher model.
- Proposed KDCTime for calibrating incorrect soft labels predicted by the teacher model, where it contains two optional strategies, i.e. KDCT and KDCR. As a consequence, KDCTime further improves the accuracy of KDTime.
- We have tested the accuracy, training time, and test time of ROCKET, InceptionTime, LSTime, KDTime, KDCTime. The results show that compared to ROCKET, KDCTime simultaneously improves the accuracy and reduces its inference time with an acceptable training time overhead. In conclusion, the performance of KDCTime is promising.

The remainder of this paper is organized as follows: The related work is reviewed in Section II. Next, LSTime, KDTime, and KDCTime are introduced in Section III. The experimental results are discussed in Section IV. Finally, Section V concludes the paper.

II. Related Work

TSC, as a traditional time-series mining research direction, has been considered as one of the most challenging problems [1]. Traditionally, 1 Nearest Neighbor (1-NN) Classifiers based on Dynamic Time Warping (DTW) distance has been shown to be a very promising approach [18]. Yet, the time complexity of DTW is unacceptable compared to Euclidean Distance (ED), i.e. $\mathcal{O}(n^2)$ compared to $\mathcal{O}(n)$. Thus, many researchers have tried to accelerate the execution time of DTW. Rakthanmanon et al. [19] proposed UCR-DTW by leveraging lower bounding and early abandoning. Sakurai et al. [20] proposed SPRING under the DTW distance, which is able to monitor time-series streams in real-time. Gong et al. [21] proposed Forward-Propagation NSPRING (FPNS) for further accelerating the speed of SPRING. Nevertheless, those researches are all concentrating on time complexity. The upper bound of their accuracy is the accuracy of DTW distance.

In order to break through the bottleneck of accuracy, researchers found that the ensembling of several classifiers could significantly improve the accuracy of TSC. Thus, Baydogan et al. [22] selected an ensemble of decision trees. Kate [23] employed an ensemble of several 1-NN classifiers with different distance measures, including DTW distance. Those methods motivated the development of an ensemble of 35 classifiers, named Collective Of Transformation-based Ensembles (COTE) [24], which ensembles those classifiers over different time-series representations instead of the same ones. After that, Lines et al. [11] extended COTE by leveraging a new hierarchical structure with probabilistic voting, called HIVE-COTE, which is currently considered the state-of-the-art approach in terms of accuracy. Nevertheless, in order to achieve such high accuracy, those methods sacrifice the training and inference time, most of which are impractical when datasets are large.

With the rapid development of deep learning, DNNs are widely applied in CV and are also increasingly employed in TSC [6]. Cui et al. [25] proposed Multi-Scale Convolutional Neural Networks (MCNN), which transforms the time-series into several feature vectors and feeds those vectors into a CNN model. Wang et al. [6] implemented several DNN models originated from CV, i.e. MultiLayer Perceptrons (MLP), Fully Convolutional Networks (FCN), and Residual Networks (ResNet), in order to test their performance in TSC, which provides a strong baseline of DNN-based approaches. Based on a more recent DNN structure, i.e. Inception module [4], Karimi-Bidhendi et al. [26] proposed an approach to transform time-series into feature maps using Gramian Angular

Difference Field (GADF), and finally feed those maps to an InceptionNet which is pre-trained for image recognition. By extending a more recent version of IncpetionNet, i.e. Inception-V4 [27], Fawaz et al. [7] proposed InceptionTime, which ensembles 5 Inception-based models to get a promising accuracy. Multi-scale Attention Convolutional Neural Network (MACNN) [28] adopted the attention mechanism to further improve the accuracy of MCNN. Instead of utilizing convolutions as parts of the model, RandOm Convolutional KErnel Transform (ROCKET) [12] employed random convolutional kernels as feature extractors converting time-series into feature vectors, which were later fed to a ridge regressor. To the best of the authors' knowledge, ROCKET owns the best accuracy and inference time balance, while other DNN-based methods always require a longer training and inference time. Actually, DNN-based methods always suffer from long training time, e.g. MACNN requires 3 days of running time for training 85 datasets from the UCR archive. That builds a huge barrier for researchers to reimplement the approach.

We found that InceptionTime is a quite competitive approach. Using only 1 InceptionTime model instead of ensembling 5 ones, it owns a slower yet acceptable training time and a 2 magnitude less inference time compared to ROCKET. Therefore, when InceptionTime only includes 1 model instead of ensembling several models, we would like to ensure the accuracy of it by the information obtained from soft labels. The idea of utilizing soft labels was proposed by Szegedy et al. [15], which controls the smooth level of soft labels by manually setting a parameter ε. Yet, a better way to determine the smooth level of soft labels is generating soft labels automatically by a teacher model, and training a student model with those labels, the idea of which comes from Knowledge Distillation (KD) [17]. Since then, many extensions of KD were proposed [29]. Some researches [30], [31] concentrated on letting the student model learn the feature maps, instead of soft labels, of the teacher model. In addition, Svitov et al. [32] leveraged the predicted labels by the teacher model as class centers, instead of soft labels, guiding the training of the student model. Oki et al. [33] integrated KD into triplet loss and utilized the predicted labels as anchor points for guiding the training of the student model. In this paper, instead of employing other types of KD methods, we still concentrated on label-based KD approaches in order to save more execution time. At last, inspired by [34], [35], the gap between the student model and the teacher model should be small, or the student model would hard to mimic the teacher model. Therefore, in this paper, a 3-layer student InceptionTime model is selected as the student model, and a 6-layer teacher model is employed as the teacher model.

III. Proposed Approaches

In this section, instead of concentrating on the model, all the proposed approaches are essentially centered on loss functions and labels. First, notations and definitions are given in Section III-A. After that, InceptionTime is briefly introduced in Section III-B. Then, Label Smoothing for InceptionTime (LSTime) is demonstrated in Section III-C. Next, Knowledge Distillation for InceptionTime (KDTime) is depicted in Section III-D. At last, Knowledge Distillation with Calibration for InceptionTime (KDCTime) is illustrated in Section III-E, where it contains 2 strategies, i.e. Calibration by Translating (CT) and Calibration by Reordering (CR).

A. Notations and Definitions

Definition 1: A time-series $\mathbf{x} \in \mathbb{R}^N$ is defined as a vector, where x_i represents the i-th value of $\mathbf{x}$. The corresponding class of $\mathbf{x}$ is a scalar $c \in \{1, 2, \ldots, C\}$, with C classes in total.

Definition 2: A class label of $\mathbf{x}$ is defined as a vector $\mathbf{y} = \{y_1, y_2, \ldots, y_C\}$, where $y_i \in [0, 1]$ represents the probability of $\mathbf{x}$ belonging to class c. In addition, Eq. (1) always holds for any $\mathbf{y}$.

$$\sum_{i=1}^{C} y_i = 1 \tag{1}$$

Definition 3: The true label of $\mathbf{x}$ is defined as an one-hot vector $\mathbf{y}^h$, where all $y_i^h = 0$ except $y_c^h = 1$, called the hard label. The equation of $\mathbf{y}^h$ is given in Eq. (2).

$$y_i^h = \begin{cases} 1, \text{if } i = c \\ 0, \text{if } i \neq c \end{cases} \tag{2}$$

Definition 4: A dataset $\mathbf{D}$ is a pair of sets, including a set of time-series $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M\}$ and a set of true labels $\mathbf{Y}^h = \{\mathbf{y}_1^h, \mathbf{y}_2^h, \ldots, \mathbf{y}_M^h\}$ respectively, where each time-series $\mathbf{x}_i$ corresponds to a true label $\mathbf{y}_i^h$.

Definition 5: An InceptionTime model is treated as a function $\mathcal{F} \in \mathbb{F}$ mapping an input $\mathbf{x}$ into an output $\mathbf{z} \in \mathbb{R}^C$, where $\mathbb{F}$ represents the hypothesis space, i.e. the space containing all possibilities of $\mathcal{F}$.

Definition 6: The predicted label $\hat{\mathbf{y}}$ is produced by normalizing $\mathbf{z}$ with the Softmax function (Eq. (3)). Thus, Eq. (1) always holds.

$$\sigma(z_i) = \hat{y}_i = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}} \tag{3}$$

Definition 7: A loss function $\mathcal{L}$ is a function measuring the difference between the predicted label $\hat{\mathbf{y}}$ and the true label $\mathbf{y}^h$, in order to determine the performance of $\mathcal{F}$.

Definition 8: The problem in the paper is defined as follows: Given a dataset $\mathbf{D}$, find an $\mathcal{F}^\star$ minimizing the predefined $\mathcal{L}$. Formally, it is demonstrated in Eq. (4).

$$\mathcal{F}^\star = \underset{\mathcal{F} \in \mathbb{F}}{\arg\min} \sum_{i=1}^{M} \mathcal{L}(\mathbf{y}_i^h, \sigma(\mathcal{F}(\mathbf{x}_i))) \tag{4}$$

To this end, important notations are briefly summarized in Table I.

TABLE I
NOTATIONS AND DEFINITIONS

Notations	Definitions
$\mathbf{x}$	A time-series
$\mathbf{y}^h$	The true label w.r.t. $\mathbf{x}$
$\mathbf{D}$	A dataset
$\mathbf{X}$	A set of time-series in $\mathbf{D}$
$\mathbf{Y}$	A set of labels in $\mathbf{D}$
$\mathcal{F}$	An InceptionTime model
$\mathbf{z}$	The output of $\mathcal{F}(\mathbf{x})$
σ	The Softmax function
$\hat{\mathbf{y}}$	The predicted label of $\sigma(\mathbf{z})$
$\mathcal{L}$	A loss function

B. InceptionTime

The ordinary Softmax Cross-Entropy Loss is adopted in InceptionTime [7]. We first implemented the one model version of InceptionTime with 3 Inception modules, denoted $\mathcal{F}_S$. Thus, the loss function of $\hat{\mathbf{y}} = \sigma(\mathcal{F}_S(\mathbf{x}))$ is given in Eq. (5).

$$\mathcal{L}_{CE}(\mathbf{y}^h, \hat{\mathbf{y}}) = -\sum_{i=1}^{C} y_i^h \log \hat{y}_i \tag{5}$$

where it is easy to know the final loss $\mathcal{L}_{CE}$ is only related to y_c^h, as all the other y_i^h are 0 (Eq. (2)). In other words, only the result of $\log \hat{y}_c$ is survived in the summation. Therefore, for simplicity, Eq. (5) can also be written as Eq. (6).

$$\mathcal{L}_{CE}(\mathbf{y}^h, \hat{\mathbf{y}}) = -\log \hat{y}_c \tag{6}$$

Note that Eq. (6) is the reason that one-hot class label $\mathbf{y}$ is called hard label, as it explicitly selects only the probability of $\mathbf{x}$ belonging class c, yet ignoring all other probabilities in the loss function. However, in a more realistic scenario, we believe such a deterministic case is rare. Hence, as also introduced in Section I, a soft version of $\mathbf{y}^h$ is more feasible in this case.

C. Label Smoothing for InceptionTime

Second, following by the assumption in Section III-B, we implemented Softmax Cross-Entropy with Label Smoothing (LS) [15], with $\mathcal{F}_S$ as the model. Therefore, the equation of label smoothed $\mathbf{y}^h$ is given in Eq. (7), denoted $\mathbf{y}^l$.

$$y_i^l = \begin{cases} (1-\varepsilon) + \varepsilon/C, & \text{if } i = c \\ \varepsilon/C, & \text{if } i \neq c \end{cases} \tag{7}$$

where ε is the smoothing coefficient set by users, representing how much the label is smoothed. Note after LS, y_i^l still satisfies Eq. (1). Alternatively, Eq. (7) can also be written as Eq. (8).

$$y_i^l = (1-\varepsilon) y_i^h + \frac{\varepsilon}{C} \tag{8}$$

As a consequence, the Softmax Cross-Entropy loss with LS is given in Eq. (9).

$$\begin{aligned} \mathcal{L}_{LS}(\mathbf{y}^l, \hat{\mathbf{y}}) &= -\sum_{i=1}^{C} y_i^l \log \hat{y}_i \\ &= -\sum_{i=1}^{C} [(1-\varepsilon) y_i^h + \frac{\varepsilon}{C}] \log \hat{y}_i \\ &= -\sum_{i=1}^{C} (1-\varepsilon) y_i^h \log \hat{y}_i - \sum_{i=1}^{C} \frac{\varepsilon}{C} \log \hat{y}_i \\ &= (1-\varepsilon)\mathcal{L}_{CE}(\mathbf{y}^h, \hat{\mathbf{y}}) + \varepsilon(-\frac{1}{C}\sum_{i=1}^{C} \log \hat{y}_i) \end{aligned} \tag{9}$$

where the left part $(1-\varepsilon)\mathcal{L}(\mathbf{y}^h, \hat{\mathbf{y}})$ represents the loss from hard labels, while the right part $\varepsilon(-\frac{1}{C}\sum_{i=1}^{C} \log \hat{y}_i)$ means the loss from soft labels. The smoothing coefficient ε controls the weights of losses from hard labels and soft labels.

Yet, manually controlling the smoothed level of labels by ε is not the best solution, since, except y_c^l, every smoothed label has the same value. Similar to hard labels, this kind of manually controlled soft labels are not practical in the real world. Therefore, generating flexible soft labels by Knowledge Distillation is then proposed.

D. Knowledge Distillation for InceptionTime

Third, we implemented Knowledge Distillation (KD) to help us generate soft labels in replacement of manually controlling them. Instead of manually setting up the soft labels, Hinton et al. [17] proposed KD for generating soft labels by a teacher model, which owns a cumbersome architecture with a large number of parameters. Intuitively, the teacher model has more potential to capture the knowledge from training data because of its scale. Next, the predicted labels from the teacher model can be regarded as the knowledge learned by it, denoted $\mathbf{y}^t$. Thus, $\mathbf{y}^t$ is an automatic soft label compared to the manual soft label $\mathbf{y}^l$. Note the one model version of InceptionTime with 6 Inception modules is incorporated as the teacher model, denoted $\mathcal{F}_T$.

In addition, instead of directly using Softmax Cross-Entropy loss, Softmax with a Temperature τ and Kullback-Leibler (KL) divergence loss are adopted. The equation of Softmax with τ is given in Eq. (10).

$$y_i^\tau = \frac{e^{z_i/\tau}}{\sum_{j=1}^{C} e^{z_j/\tau}} \tag{10}$$

where the Temperature τ is a parameter for fine-tuning the smoothed level of predicted labels $\mathbf{y}^t$ from the teacher model and $\hat{\mathbf{y}}$ from the student model, denoted $\mathbf{y}^{t_\tau}$ and $\hat{\mathbf{y}}^\tau$. Note $\tau = 1$ means the labels keep unchanged, while $\tau < 1$ or $\tau > 1$ represents the labels are steeper or smoother respectively. For an extreme example, if $\tau \to \infty$, we will have $\forall i, y_i^\tau = 1/C$. To this end, the loss of $\mathbf{y}^{t_\tau}$ and $\hat{\mathbf{y}}^\tau$ can be measured by KL divergence, the equation of which is given in Eq. (11).

$$\mathcal{L}_{KL}(\mathbf{y}^{t_\tau}, \hat{\mathbf{y}}^\tau) = \sum_{i=1}^{C} y_i^{t_\tau} \log \frac{y_i^{t_\tau}}{\hat{y}_i^\tau} \tag{11}$$

After that, $\mathcal{L}_{KL}(\mathbf{y}^{t_\tau}, \hat{\mathbf{y}}^\tau)$ representing the loss of soft labels and $\mathcal{L}_{CE}(\mathbf{y}^h, \hat{\mathbf{y}})$ representing the loss of hard labels are incorporated into a whole for training the student model, which has a relatively small architecture with less parameters. This procedure is regarded as distilling the knowledge from a teacher model into a student model, called KD, in order to preserve the accuracy of the teacher model while reducing its time and space complexity. Note $\mathcal{F}_S$ is selected as the student model, which is the one model version of InceptionTime with 3 Inception modules. The equation of KD loss is given in Eq. (12).

$$\begin{aligned}&\mathcal{L}_{KD}(\mathbf{y}^h, \hat{\mathbf{y}}, \mathbf{y}^{t_\tau}, \hat{\mathbf{y}}^\tau)\\&= (1-\varepsilon)\mathcal{L}_{CE}(\mathbf{y}^h, \hat{\mathbf{y}}) + \varepsilon\tau^2\mathcal{L}_{KL}(\mathbf{y}^{t_\tau}, \hat{\mathbf{y}}^\tau)\end{aligned} \tag{12}$$

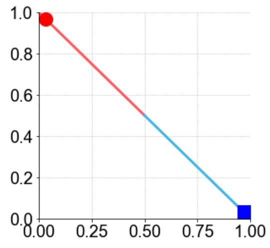
where ε controls the weight of $\mathcal{L}_{CE}(\mathbf{y}^h, \hat{\mathbf{y}})$ (Eq. (5)) and $\mathcal{L}_{KL}(\mathbf{y}^{t_\tau}, \hat{\mathbf{y}}^\tau)$ (Eq. (11)). Note τ^2 is necessary because the scale of $\mathcal{L}_{KL}(\mathbf{y}^{t_\tau}, \hat{\mathbf{y}}^\tau)$ becomes smaller after fine-tuning by τ. Thus, multiplying a τ^2 helps it to be the same scale of $\mathcal{L}_{CE}(\mathbf{y}^h, \hat{\mathbf{y}})$, so that the total loss has no preference to $\mathcal{L}_{CE}(\mathbf{y}^h, \hat{\mathbf{y}})$.

Nevertheless, similar to teachers in real life, the teacher model is not ensured to be 100% correct. Sometimes it may misguide the student model to wrong answers. To be specific, the incorrect soft labels generated by the teacher model will also result in the wrong labels predicted by the student model. In order to alleviate the affection of incorrect labels, KD adopts $\mathcal{L}_{CE}(\mathbf{y}^h, \hat{\mathbf{y}})$ and τ. Nonetheless, that brings additional hyperparameters into the model. Therefore, we would like to propose a better method to alleviate the affection of incorrect labels while not bringing additional hyperparameters.

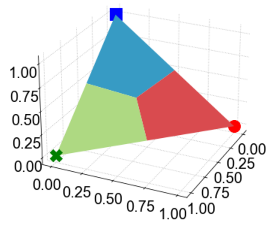
E. Knowledge Distillation with Calibration for InceptionTime

At last, we propose Knowledge Distillation with Calibration (KDC) to calibrate the incorrect soft labels generated by the teacher model before distillation. Note the teacher model and student model are $\mathcal{F}_T$ and $\mathcal{F}_S$ respectively (Section III-D). In this way, it is not necessary to employ $\mathcal{L}_{CE}(\mathbf{y}, \hat{\mathbf{y}})$ and τ in KDC. In order to calibrate the incorrect soft labels, all labels $\mathbf{y}$ are regarded as vectors geometrically, including the hard label $\mathbf{y}^h$ and the soft label $\mathbf{y}^t$ generated by the teacher model. From this point of view, according to Eq. (1), the feasible solution space of $\mathbf{y}$ is a triangular hyperplane, named the label space. In other words, all $\mathbf{y}$ are located on a triangular hyperplane. Fig. 3(a) gives an example when $C = 2$. In this case, the triangular hyperplane is an 1-D line in 2-D space. Next, Fig. 3(b) shows another example when $C = 3$. In this case, the triangular hyperplane is a 2-D regular triangle in 3-D space. Last, the triangular hyperplane is a 3-D regular tetrahedron in 4-D space when $C = 4$. Nonetheless, we failed to plot a 4-D space in figures. Note that distinct colors represent the areas of distinct classes. Thus, it is potential to calibrate $\mathbf{y}^t$ from its original position to the target position $\mathbf{y}^h$, if $\mathbf{y}^t$ is located in the wrong area. In addition, all $\mathbf{y}^h$ are located at the vertices of the triangular hyperplane, as marked in Fig. 3(a) and Fig. 3(b).

Therefore, our main task is to propose a proper method in order to modify $\mathbf{y}^t$ to its correct area while its new



(a) The 2-D label space when $C = 2$



(b) The 3-D label space when $C = 3$

Fig. 3. Two examples showing the label spaces when $C = 2$ and $C = 3$

position is located between $\mathbf{y}^t$ and $\mathbf{y}^h$. The calibrated $\mathbf{y}^t$ is denoted as $\mathbf{y}^{t_\varsigma}$. To this end, two approaches for calibration are proposed, which are calibration by translating and calibration by reordering. Note that only incorrect predicted labels will be calibrated. Formally, given a $\mathbf{y}^t$ and its corresponding $\mathbf{y}^h$, $\mathbf{y}^{t_\varsigma}$ will be computed only when $\arg\max_i\{y_i^t\} \neq \arg\max_i\{y_i^h\}$. In other words, $\arg\max_i\{y_i^t\} \neq c$.

1) Calibration by Translating: Calibration by Translating (CT) represents geometrically translate $\mathbf{y}^t$ from its original position to $\mathbf{y}^h$, the equation of which is shown in Eq. (13).

$$\mathbf{y}^{t_\varsigma} = \mathbf{y}^t + \omega(\mathbf{y}^h - \mathbf{y}^t) \tag{13}$$

where $(\mathbf{y}^h - \mathbf{y}^t)$ represents the vector from $\mathbf{y}^t$ to $\mathbf{y}^h$, while $\omega \in [0, 1]$ is a calibration coefficient controlling the distance $\mathbf{y}^t$ moves towards $\mathbf{y}^h$. It is easy to know that $\mathbf{y}^{t_\varsigma} = \mathbf{y}^h$ when $\omega = 1$, and $\mathbf{y}^{t_\varsigma} = \mathbf{y}^t$ when $\omega = 0$. In this case, it is simply substitution instead of calibration.

Hence, ω is the key coefficient defining the degree of calibration. We define the equation of ω as Eq. (14).

$$\omega = \frac{\delta}{\|\mathbf{y}^h - \mathbf{y}^t\|_2} \tag{14}$$

where δ is the minimum distance between $\mathbf{y}^h$ and $\mathbf{y}^t$ when $\arg\max_i\{y_i^t\} \neq \arg\max_i\{y_i^h\}$, and $\|\mathbf{y}^h - \mathbf{y}^t\|_2$ is the current distance between $\mathbf{y}^h$ and $\mathbf{y}^t$. It is easy to know that $\|\mathbf{y}^h - \mathbf{y}^t\|_2 \geq \delta$ always holds.

To this end, we calculated δ, and got the magic number $\delta = 1/\sqrt{2}$. The procedure of calculation is given in Appendix A. As a consequence, Eq. (13) can be also rewritten as (15).

$$\mathbf{y}^{t_\varsigma} = \mathbf{y}^t + \frac{1}{\sqrt{2}}\frac{\mathbf{y}^h - \mathbf{y}^t}{\|\mathbf{y}^h - \mathbf{y}^t\|_2} \tag{15}$$

where we know the unit vector $(\mathbf{y}^h - \mathbf{y}^t)/\|\mathbf{y}^h - \mathbf{y}^t\|_2$ decides the direction of translating, while $\delta = 1/\sqrt{2}$ determines the distance of translating. In this way, it ensures that all $\mathbf{y}^{t_\varsigma}$ stay in the label space, since it guarantees $\omega \leq 1$. In addition, it also guarantees that $\omega \geq 0$, which leads to a consequence that $\mathbf{y}^t$ will not stay unchanged.

2) Calibration by Reordering: Calibration by Reordering (CR) represents reprioritizing the values of the incorrect predicted label based on a specific strategy. To be concrete, given a $\mathbf{y}^t$ and its corresponding $\mathbf{y}^h$, some y_i^t will be resorted if $\arg\max_i\{y_i^t\} \neq \arg\max_i\{y_i^h\}$. Therefore, our main task is to design a reordering strategy.

Algorithm 1 Algorithm of calibration by reordering

```
Input: Given a $\mathbf{y}^t$ and its corresponding $\mathbf{y}^h$
Output: The reordered label $\mathbf{y}^{t_\varsigma}$
1: Sort $\mathbf{y}^t$ to get $\mathbf{y}^{t_s} = \{y^t_{1st}, y^t_{2nd}, \cdots y^t_{Cth}\}$
2: $y^t_{tmp} \leftarrow y^t_{1st}$
3: for $ith = 1st, 2nd, \ldots,$ until $ith = c$ do
4:     $y^t_{ith} \leftarrow y^t_{(i+1)th}$
5: $y^t_c \leftarrow y^t_{tmp}$
```

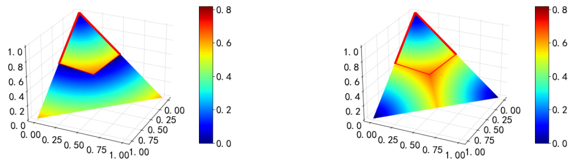
Given a $\mathbf{y}^t$ awaiting for reordering. The reordering strategy is designed as follows: 1) $\mathbf{y}^t$ is sorted by descending order. The sorted $\mathbf{y}^t$ is denoted $\mathbf{y}^{t_s}$. Thus, since the descending order of y^t_i is random, we have $\mathbf{y}^{t_s} = \{y^{t_s}_1, y^{t_s}_2, \ldots, y^{t_s}_C\} = \{y^t_{1st}, y^t_{2nd}, \cdots y^t_{Cth}\}$, where y^t_{1st} is the largest y^t_i, y^t_{2nd} is the second largest y^t_i, and so on and so forth. Note $\mathbf{y}^{t_s}$ is in descending order, which means $y^{t_s}_1 = y^t_{1st} = \max_i\{y^t_i\}$. 2) After defining a temporary value $y^t_{tmp} = y^t_{1st}$, the value of $y^t_{(i+1)th}$ will be assigned to y^t_{ith}, from y^t_{1st}, y^t_{2nd}, all the way to $y^t_{ith} = y^t_c$. 3) Assign the value of y^t_{tmp} to y^t_c.

The algorithm of the whole procedure is given in Algorithm 1. It ensures that $\mathbf{y}^{t_\varsigma}$ is located in the label space, since $\mathbf{y}^{t_\varsigma}$ is only the reordered version of $\mathbf{y}^t$. In addition, it guarantees that $\mathbf{y}^{t_\varsigma}$ is geometrically located in its class area. In other words, $\arg\max_i\{y^{t_\varsigma}_i\} = \arg\max_i\{y^h_i\}$. As a consequence, $\mathbf{y}^t$ is successfully calibrated by reordering.

3) Analysis of CT and CR: We theoretically analyzed the difference between CT and CR in terms of calibration. To be specific, for one $\mathbf{y}^t$ that is not located in the correct area in the label space, KDC will be utilized to calibrate $\mathbf{y}^t$ as $\mathbf{y}^{t_\varsigma}$ either by CT or CR, where $\mathbf{y}^{t_\varsigma}$ is located in the correct area. It can be concluded that there must exist a mapping of any label from the incorrect area to the correct area in the label space. Thus, we wish to know the mapping relationship by calculating the distance between $\mathbf{y}^h$ and $\mathbf{y}^t$ after KDC.

Fig. 4 shows an example in 3-D space, where the color map of the distance between $\mathbf{y}^h$ and $\mathbf{y}^t$ after KDC in the label space is illustrated. The distance calculating function is defined as $\mathcal{D}(\mathbf{y}^h, \mathbf{y}^t) = \|\mathbf{y}^h - \mathbf{y}^t\|_2$. As $\mathbf{y}^t$ in the red area (Correct area) do not require calibration, their distance is calculated by $\mathcal{D}(\mathbf{y}^h, \mathbf{y}^t)$ directly. On the contrary, $\mathbf{y}^t$ out the red area (Incorrect area) are calibrated as $\mathbf{y}^{t_\varsigma}$. Therefore, their distance is calculated by $\mathcal{D}(\mathbf{y}^h, \mathbf{y}^{t_\varsigma})$. As shown in Fig. 4(a), the $\mathbf{y}^t$ being close to the edge of the red area are mapped close to the vertex, i.e. the hard label $\mathbf{y}^h$, while the $\mathbf{y}^t$ being far away from the red area are mapped to the edge of the red area. Yet in Fig. 4(b), the $\mathbf{y}^t$ being close to the edge of the red area are mapped also close to the edge, while the $\mathbf{y}^t$ being far away from the red area are mapped to close to $\mathbf{y}^h$. As a consequence, CT keeps the relative position of incorrect $\mathbf{y}^t$ unchanged, which means it preserves the spatial information of $\mathbf{y}^t$. In addition, CR keeps the shape of the distribution of incorrect $\mathbf{y}^t$ unchanged, which means it preserves the distributional information of $\mathbf{y}^t$.

To this end, we are in the position to define the loss function of KDC. Unlike the loss function of KD, we employ KL-divergence only without temperature τ and the Cross-Entropy part, as shown in Eq. 16.



(a) Calibration by traslating (b) Calibration by reordering

Fig. 4. A 3-D example illustrating the color map of the distance between $\mathbf{y}^h$ and $\mathbf{y}^t$ after KDC in the label space

Algorithm 2 Algorithm of KDCTime

```
Input: The training data $\mathbf{X}$ and its corresponding labels $\mathbf{Y}$.
Output: The trained student model $\mathcal{F}^*_S$
1: Initialize a teacher model $\mathcal{F}_T$
2: Train $\mathcal{F}_T$ by $\mathbf{X}$ and $\mathbf{Y}$ to get $\mathcal{F}^*_T$
3: Generate $\mathbf{Y}^t$ by $\mathcal{F}^*_T$
4: for each $\mathbf{y}^t_i$ do
5:     if $\mathbf{y}^t_i$ and $\mathbf{y}^h_i$ belong to distinct class then
6:         Calibrate $\mathbf{y}^t_i$ to get $\mathbf{y}^{t_\varsigma}_i$    ▷ Eq. 15 or Algorithm 1
7: Initialize the student model $\mathcal{F}_S$
8: Train $\mathcal{F}_S$ by $\mathbf{X}$ and $\mathbf{Y}^{t_\varsigma}$ to get $\mathcal{F}^*_S$
```

TABLE II
PROBLEMATIC DATASETS

Dataset Name		
AllGestureWiimoteX	DodgerLoopDay	GestureMidAirD1
AllGestureWiimoteY	DodgerLoopGame	GestureMidAirD2
AllGestureWiimoteZ	DodgerLoopWeekend	GestureMidAirD3
MelbournePedestrian	PickupGestureWiimoteZ	GesturePebbleZ1
ShakeGestureWiimoteZ	PLAID	GesturePebbleZ2

$$\mathcal{L}_{KDC}(\mathbf{y}^{t_\varsigma}, \hat{\mathbf{y}}) = \mathcal{L}_{KL}(\mathbf{y}^{t_\varsigma}, \hat{\mathbf{y}}) = \sum_{i=1}^{C} y^{t_\varsigma}_i \log \frac{y^{t_\varsigma}_i}{\hat{y}_i} \tag{16}$$

where $\mathbf{y}^{t_\varsigma}$ is the calibrated label generated by the teacher model, while $\hat{\mathbf{y}}$ is the label generated by the student model. Compared to $\mathcal{L}_{KD}(\mathbf{y}^h, \hat{\mathbf{y}}, \mathbf{y}^{t_\tau}, \hat{\mathbf{y}}^\tau)$, $\mathcal{L}_{KDC}(\mathbf{y}^{t_\varsigma}, \hat{\mathbf{y}})$ does not contains any hyperparameter and computes only KL divergence, which reduces much computational time and the complexity of hyperparameter tuning.

Thus, the total process of KDC can be concluded as 3 steps: 1) Train a teacher model with a heavier and more complex architecture by true labels (Hard labels). Generate the predicted labels by the teacher model; 2) Calibrate the incorrectly predicted labels; 3) Train the student model with a relatively small and simple architecture by calibrated labels (Soft labels). The algorithm of KDC is given in Algorithm 2.

IV. EXPERIMENTS

We conduct the experiments on UCR datasets, in which there are 128 datasets. Yet, there are 15 problematic datasets with NaN (Not a Number) values, since missing data or various time-series lengths. They are listed in Table II. Therefore, the remaining 113 datasets are selected for experiments.

In the experiments, ROCKET, Softmax Cross-Entropy for InceptionTime (ITime), label smoothing for InceptionTime

(LSTime), Knowledge Distillation for InceptionTime (KDTime), and KD with calibration for InceptionTime (KDCTime) are compared, where KDCTime includes two calibrating methods, i.e. KDC by translating (KDCT) and KDC by reordering (KDCR) respectively. Except for ROCKET, all the aforementioned methods can be concluded as ITime-based approaches. At first, in order to find the best hyperparameter for LSTime, KDTime, KDCT, and KDCR, we conducted the experiments of hyperparameter study for those approaches. Note ITime does not have any hyperparameter to be tuned. After that, we tested the accuracy, training time, and test time of the aforementioned approaches.

Similar to [7], critical difference diagrams are drawn in the paper for better illustrating the results of different approaches, as the results of 113 datasets are hard to be depicted clearly. Critical difference diagram is a diagram drawn by the following steps: 1) Execute the Friedman test [36] for rejecting the null hypothesis. 2) Perform the pairwise post-hoc analysis [37] by a Wilcoxon signed-rank test with Holm's alpha (5%) correction [38]. 3) Visualize the statistical result by [39], where a thick horizontal line represents that the approaches are not significantly different with respect to results.

Our experiments are conducted on a computer equipped with an Intel Core i9-11900 CPU at 2.50GHz, 32GB memory, and an NVIDIA GeForce RTX 3090 GPU. The operating system is Windows 10. Additionally, the development environment is Anaconda 4.10.3 with Python 3.8.8 and Pytorch 1.9.0.

A. Hyperparameter Study for ITime-based Approaches

In this section, we first searched 3 hyperparameters, i.e. batch size, epoch, and learning rate, on ITime. Thus, those hyperparameters of LSTime, KDTime, KDCT, and KDCR can be determined also, since all these methods are based on InceptionTime. After that, ε in LSTime, ε and τ in KDTime, and ε in KDCT and KDCR were searched separately.

1) Batch Size: First, the batch size was set to 64 without searching. The reason is that batch size is theoretically larger the better. An extreme case is the full-batch. However, in deep learning tasks, that always leads to a consequence where graphics memory on GPU is infeasible to load the large dataset. Additionally, The batch size and the epoch are depending on each other, i.e. the more batch size represents the more epochs for converging, which means a longer training time. Thus, the batch size is always empirically set to either 16, 32, 64, 128, or 256. Hence, 64 was adopted in the paper.

2) Epoch: With a fixed batch size 64, we compared the accuracy of 5 different epochs, which were 64, 128, 256, 512, and 1024 respectively. The critical difference diagram is given in Fig. 5, where 1024 epochs own the best accuracy. Yet, 1024 epochs are of no critical difference with 512 epochs. Since the training time of DNN-based approaches is slow, 512 is selected as the epoch in order to save the training time. In addition, we also employed the early stop strategy with patience equals to 80 epochs for reducing the training time and alleviating overfitting.

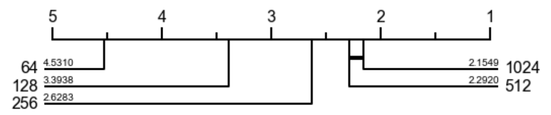


Fig. 5. Critical difference diagram for different epochs

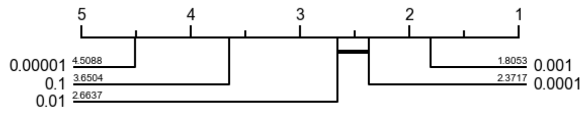


Fig. 6. Critical difference diagram for different learning rates

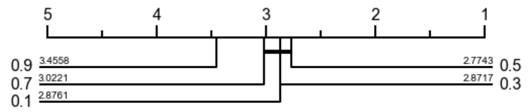


Fig. 7. Critical difference diagram for different ε in LSTime

3) Learning Rate: The accuracy of 5 distinct learning rates were tested in total, which were 0.1, 0.01, 0.001, 0.0001, and 0.00001. Moreover, learning rate decay was employed in order to stabilize the training process. We leveraged fixed-step decay, also called piecewise constant decay, as the decay strategy, where the step size is set as 35 and gamma is set as 0.5. In other words, the learning rate will multiply by 0.5 for every 35 epochs. Note there are 256 epochs, which ensures 7 times of decay in total. The critical difference diagram is shown in Fig. 6, where the learning rate being equal to 0.01 has the highest accuracy. Thus, 0.01 is employed as the learning rate in the paper.

4) ε in LSTime: The smoothing coefficient ε (Eq. (9)) represents the smoothed level of labels. We tested the accuracy of 5 different ε in LSTime, which are 0.1, 0.3, 0.5, 0.7, and 0.9. After all, the critical difference diagram is given in Fig. 7, where 0.5 gets the best accuracy. This claims that ε should not be too small or too big, since a small ε gives the label little additional information, while a big ε causes too much information loss from its original class. In addition, the accuracy of big ε is less than that of small ε, which means information from its original class is important, and it is not a good idea to completely abandon that information.

5) ε and τ in KDTime: KDTime contains 2 hyperparameters, which are ε and τ (Eq. (12)), where ε controls the weight of losses between the hard label and the smooth label respectively, and τ fine-tunes the smoothed level of smooth labels (Predicted labels by teacher model). Thus, we tested the accuracy of 5 distinct ε in KDTime, which are 0.1, 0.3, 0.5, 0.7, and 0.9, while we also compared the accuracy of 5 different τ, which are 2, 4, 8, 16, and 32. The critical diagrams of ε and τ are shown in Fig. 8 and Fig. 9. Fig. 8 shows that $\varepsilon = 0.5$ has the best accuracy. The ε close to 0 or 1 will reduce the accuracy. In addition, Fig. 9 shows that $\tau = 8$ is the best. Similarly, the accuracy of KDTime will decrease if τ is too big or too small.

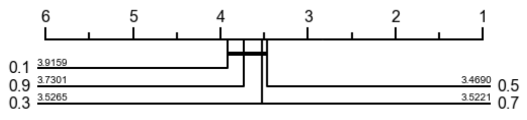


Fig. 8. Critical difference diagram for different ε in KDTime

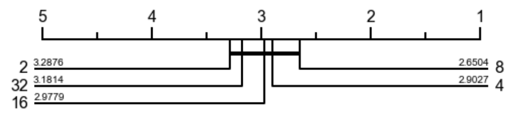


Fig. 9. Critical difference diagram for different τ in KDTime

To this end, we are able to conclude that all the ITime-based methods select 64 as the batch size, 512 as the epoch, and 0.01 as the learning rate. In addition, ε in LSTime is set to 0.5, ε and τ in KDTime is set to 0.5 and 8 respectively. Finally, Adam is adopted as the optimizing algorithm in order to update the model.

B. Accuracy of Different Approaches

In this section, we compared the accuracy of ROCKET, ITime, LSTime, KDTime, KDCTime by Translating (KDCT), and KDCTime by Reordering (KDCR). The accuracy of all approaches is averaged from 10 times running. In the meantime, the standard deviation of these 10 accuracies is also calculated. The result is listed in Table III (Appendix B). The number before the $\pm$ sign represents accuracy, while the number after the $\pm$ sign means standard deviation. Besides, the bold number represents the best accuracy or standard deviation among all approaches.

As shown in Table III, KDCR gets the highest accuracy on 54 datasets, which claims that its accuracy is competitive and promising. Yet, its converging process is not as stable as ROCKET, since ROCKET has the lowest standard deviation on the majority of datasets. Besides, the critical difference diagram is also given for better illustrating the result of Table III, which is shown in Fig. 10. Note that Fig. 10 also demonstrates the accuracy of the teacher model used in KDTime, KDCT, and KDCR, which is an InceptionTime model with 6 Inception modules. It is also trained 10 times and the best one is selected as the teacher. That is the reason why its accuracy is the best one. In addition, KDCR and KDCT are of no significant difference with KDTime. The reason is that the accuracy of 75 datasets are more than 0.8 for the teacher model, which claims the majority of datasets meets the bottleneck of improving accuracy. In other words, only a small number of samples can be calibrated by KDCTime. Thus, We claim that the results of those 85 datasets dominate the other 28 datasets in the critical difference diagram. By ignoring that part of the results, the accuracy of KDCR and KDCT is more promising, as shown in Fig. 11.

In summary, KDCR owns the best accuracy, which is better than KDCT. The reason is that KDCR keeps the information from marginal labels. In Fig. 3, marginal labels represent the

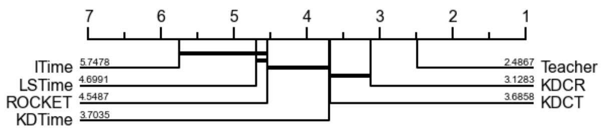


Fig. 10. Critical difference diagram illustrating the accuracy of different approaches

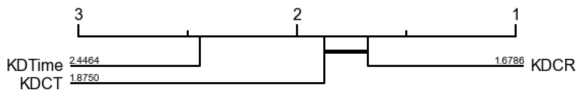


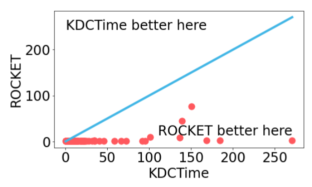
Fig. 11. Critical difference diagram illustrating the accuracy of three approaches on datasets with accuracy less than 0.8 for KDTime

labels located in the middle area of the label space. In Fig. 4(b), we know the labels located in the middle area have a long distance to $\mathbf{y}^h$, where they do not deterministically belong to any class, meaning that they contain abundant information from other classes. Yet, KDCT calibrates the marginal labels close to $\mathbf{y}^h$, which losses that information. In addition, KDCR calibrates the labels close to other classes as close to the correct one, as shown in Fig. 4(b), which eliminates the misguiding from deterministic incorrect labels. Nevertheless, KDCT keeps the information of those labels from incorrect classes.

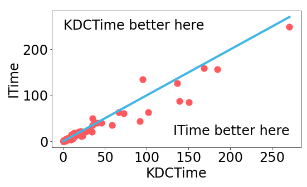
C. Training and Test Time of Different Approaches

In this section, we compared the training and test time of ROCKET, ITime, LSTime, KDTime, and KDCTIme, ignoring KDCT or KDCR, as their training and inference time are of no significant difference. Instead of showing all the results in a table, diagrams of comparison were selected for better illustrating the results.

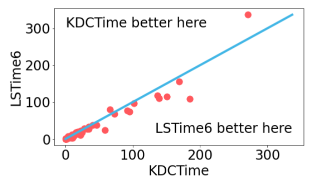
Fig. 12 demonstrates the training time of KDCTime compared with ROCKET, ITime, LSTime, and KDTime. In Fig. 12(a), it shows that KDCTime is much slower. In detail, the training time of KDCTime on 77 datasets is below 1 order of magnitude slower than ROCKET, while that on 36 datasets is more than 1 order of magnitude slower. That is because all ITime-based approaches, including KDCTime, employ Gradient Descent as the optimizing algorithm, which requires the inference on many training samples for each update and a large number of updates in total, e.g. 64 samples for each update, many times of updating in 1 epoch, and 512 epochs in total. On the contrary, ROCKET utilizes the ridge classifier and solves the ridge regression problem directly. However, the training time of KDCTime is still acceptable, which requires around 1 hour to train 113 UCR datasets and 10 hours for 10 times running in total. Besides, as shown in Fig. 12(b) and (c), the training time of ITime and LSTime is similar to KDCTime. The reason is that their model is the same, which is InceptionTime with 3 Inception modules. Still, KDCTime needs an extra teacher model in order to guild the training of its student model, which leads to a consequence that KDCTime requires extra training time for the



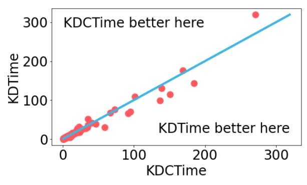
(a) KDCTime and ROCKET



(b) KDCTime and ITime

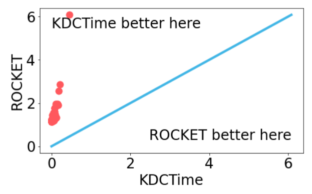


(c) KDCTime and LSTime

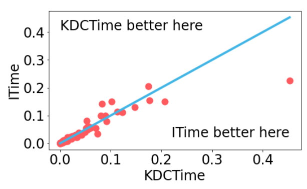


(d) KDCTime and KDTime

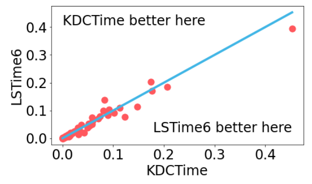
Fig. 12. The training time of KDCTime compared with ROCKET, ITime, LSTime, and KDTime



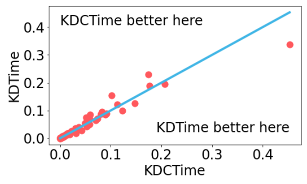
(a) KDCTime and ROCKET



(b) KDCTime and ITime



(c) KDCTime and LSTime



(d) KDCTime and KDTime

Fig. 13. The test time of KDCTime compared with ROCKET, ITime, LSTime, and KDTime

teacher model. Note that the training of the teacher model is required for only one time. Once the teacher model is obtained, that model is available for multiple times of training for the student model. Fig. 12(d) shows that the training time of KDCTime is of no significant difference with KDTime, as their models are the same, they both require the teacher model, and Gradient Descent is selected as their optimizing algorithms. In conclusion, the difference of training time mainly appears in 3 categories of methods, where the first one is ROCKET, the second one is ITime-based methods without KD, and the last one is ITime-based approaches with KD.

Fig. 13 demonstrates the test time of KDCTime compared with ROCKET, ITime, LSTime, and KDTime. In Fig. 13(a), it shows that KDCTime is much faster than ROCKET. To be specific, the test time of KDCTime on 42 datasets is 1 order of magnitude faster than ROCKET, that on 61 datasets is 2 orders of magnitude faster, and that on 10 datasets is 3 orders of magnitude faster. That is because in the stage of test, without the computational time of Gradient Descent, KDCTime only requires the time of inference on test samples, which is the same as ROCKET. In this scenario, the computational time of 10000 random convolutional kernels in ROCKET is much slower than the 3 Inception modules in KDCTime. In addition, as shown in Fig. 13(b), (c), and (d), the test time of those approaches are of no difference with KDCTime, since their inference models are all InceptionTime with 3 Inception modules. As a consequence, the difference of test time can be categorized as 2 groups, which are ROCKET and ITime-based approaches, regardless of ITime-based approaches with or without KD.

V. Conclusion

In this paper, we discovered the DNN-based TSC approaches are easy to be overfitting on the UCR datasets, which is caused by the few-shot problem in the UCR archive. Thus, in order to alleviate overfitting, Label Smoothing for InceptionTime (LSTime) was first proposed by utilizing soft labels. Next, instead of manually adjusting soft labels, Knowledge Distillation for InceptionTime (KDTime) was proposed in order to automatically generate soft labels. At last, in order to rectify the incorrect predicted soft labels from the teacher model, KD with calibration (KDC) was proposed, where it has two optional strategies, namely KDC by Translating (KDCT) and KDC by Reordering (KDCR).

The experimental results show that the accuracy of KDCT and KDCR is promising, while KDCR gets the highest one. In addition, including KDCT and KDCR, all InceptionTime-based (ITime-based) approaches are 2 orders of magnitude faster than ROCKET on test time, since the ITime model is the majority factor for the inference time. The training time of ITime-based approaches is slower than ROCKET, yet it is in an acceptable range and worthwhile in order to obtain a promising accuracy and fast inference time. At last, KDCT and KDCR do not introduce any additional hyperparameter compared to ITime.

In the future, instead of just concentrating on the loss functions and labels, we will try various models, in order to propose a brand new model which owns a high generalization capability.

Appendix A
The Procedure to Calculate δ

Given a hard label $\mathbf{y}^h$ and a soft label $\mathbf{y}^t$, where they both satisfy $\sum_{i=1}^{C} y_i = 1$ and $\forall i, y_i \geq 0$. By treating $\mathbf{y}^h$ and $\mathbf{y}^t$ as vectors, we want to find the minimum distance between them when $\arg\max_i\{y_i^t\} \neq \arg\max_i\{y_i^h\}$. Let $c = \arg\max_i\{y_i^h\}$, so that $y_c^h = 1$ and $c \neq \arg\max_i\{y_i^t\}$. Let $m = \arg\max_i\{y_i^t\}$, so that $\forall i, y_i^t \leq y_m^t$. Therefore, we have the following optimization objective:

$$\begin{aligned}
&\min \|\mathbf{y}^h - \mathbf{y}^t\|_2 \\
&\text{s.t.} \sum_{i=1}^{C} y_i^t = 1 \\
&\qquad y_i^t \geq 0, i = 1, 2, \ldots, C \\
&\qquad y_i^t \leq y_m^t, i = 1, 2, \ldots, C
\end{aligned}$$

where we know $y_c^h = 1$ and $\forall i \neq c, y_i^h = 0$. Thus, $\min \|\mathbf{y}^h - \mathbf{y}^t\|_2 = \min\{(1 - y_c^t)^2 + \sum_{i \neq c}(y_i^t)^2\}$. Since $\sum_{i=1}^{C} y_i^t = 1$, we let $y_c^t = 1 - \sum_{i \neq c} y_i^t$. Thus, $\min\{(1-y_c^t)^2 + \sum_{i\neq c}(y_i^t)^2\} = \min\{(\sum_{i\neq c} y_i^t)^2 + \sum_{i\neq c}(y_i^t)^2\}$. In this way, our optimization objective can be rewritten as follows:

$$\begin{aligned} &\min\{(\sum_{i\neq c} y_i^t)^2 + \sum_{i\neq c}(y_i^t)^2\} \\ &\text{s.t.} - y_i^t \leq 0, i = 1, 2, \ldots, C \\ &\quad y_i^t - y_m^t \leq 0, i = 1, 2, \ldots, C \end{aligned}$$

This is an optimization problem with inequality constraints. Therefore, we can define its Lagrangian function as:

$$\begin{aligned} &\mathcal{L}(\mathbf{y}^t, \boldsymbol{\lambda}, \boldsymbol{\mu}) \\ &= (\sum_{i\neq c} y_i^t)^2 + \sum_{i\neq c}(y_i^t)^2 - \sum_i \lambda_i y_i^t + \sum_i \mu_i (y_i^t - y_m^t) \end{aligned}$$

where $\boldsymbol{\lambda} \in \mathbb{R}^C$ and $\boldsymbol{\mu} \in \mathbb{R}^C$ are two sets of Lagrangian multipliers. By adopting Karush-Kuhn-Tucker (KKT) Conditions, we have:

$$\begin{cases} -\lambda_c + \mu_c = 0, i = c \\ 2\sum_{j\neq c} y_j^t + 2y_m^t - \lambda_m - \sum_{j\neq m}\mu_j = 0, i = m \\ 2\sum_{j\neq c} y_j^t + 2y_i^t - \lambda_i + \mu_i = 0, i \neq c \ and \ i \neq m \\ -\lambda_i y_i^t = 0, i = 0, 1, \ldots, C \\ \mu_i(y_i^t - y_m^t) = 0, i = 0, 1, \ldots, C \\ \lambda_i \geq 0, i = 0, 1, \ldots, C \\ \mu_i \geq 0, i = 0, 1, \ldots, C \end{cases}$$

At last, after solving this system of equations, we know $\mathcal{L}(\mathbf{y}^t, \boldsymbol{\lambda}, \boldsymbol{\mu})$ obtains the minimum value when $y_c^t = 1/2$, $y_m^t = 1/2$, and all other $y_i^t = 0$. As a result, δ can be calculated as follows:

$$\delta = \min \|\mathbf{y}^h - \mathbf{y}^t\|_2 = \sqrt{(1-\frac{1}{2})^2 + (-\frac{1}{2})^2} = \frac{1}{\sqrt{2}}$$

Appendix B
The accuracy of different approaches

The accuracy of different approaches on UCR datasets is given in Table III on the last page.

References

[1] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P. Muller, "Deep learning for time series classification: a review," *Data Mining and Knowledge Discovery*, vol. 33, no. 4, pp. 917–963, 2019.

[2] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu, "Squeeze-and-excitation networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 8, pp. 2011–2023, 2020.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[4] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1–9.

[5] W. V. Anthony Bagnall, Jason Lines and E. Keogh, "The uea & ucr time series classification repository," 2018, uRL: http://www.timeseriesclassification.com/.

[6] Z. Wang, W. Yan, and T. Oates, "Time series classification from scratch with deep neural networks: A strong baseline," in *International Joint Conference on Neural Networks*, 2017, pp. 1578–1585.

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

[8] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[9] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009, uRL: https://www.cs.utoronto.ca/∼kriz/cifar.html.

[10] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[11] J. Lines, S. Taylor, and A. J. Bagnall, "Time series classification with HIVE-COTE: the hierarchical vote collective of transformation-based ensembles," *ACM Transactions on Knowledge Discovery from Data*, vol. 12, no. 5, pp. 52:1–52:35, 2018.

[12] 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.

[13] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, 2015, pp. 448–456.

[14] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[15] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2818–2826.

[16] Y. Wan and Y. Si, "A formal approach to chart patterns classification in financial time series," *Information Sciences*, vol. 411, pp. 151–175, 2017.

[17] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[18] A. J. Bagnall, J. Lines, A. Bostrom, J. Large, and E. J. 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] T. Rakthanmanon, B. J. L. Campana, A. Mueen, G. E. A. P. A. Batista, M. B. Westover, Q. Zhu, J. Zakaria, and E. J. Keogh, "Searching and mining trillions of time series subsequences under dynamic time warping," in *International Conference on Knowledge Discovery and Data Mining*, 2012, pp. 262–270.

[20] Y. Sakurai, C. Faloutsos, and M. Yamamuro, "Stream monitoring under the time warping distance," in *International Conference on Data Engineering*, 2007, pp. 1046–1055.

[21] X. Gong, S. Fong, and Y. Si, "Fast multi-subsequence monitoring on streaming time-series based on forward-propagation," *Information Sciences*, vol. 450, pp. 73–88, 2018.

[22] M. G. Baydogan, G. C. Runger, and E. Tuv, "A bag-of-features framework to classify time series," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 11, pp. 2796–2802, 2013.

[23] R. J. Kate, "Using dynamic time warping distances as features for improved time series classification," *Data Mining and Knowledge Discovery*, vol. 30, no. 2, pp. 283–312, 2016.

[24] A. J. Bagnall, J. Lines, J. Hills, and A. Bostrom, "Time-series classification with COTE: the collective of transformation-based ensembles," in *International Conference on Data Engineering*, 2016, pp. 1548–1549.

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

[26] S. Karimi-Bidhendi, F. Munshi, and A. Munshi, "Scalable classification of univariate and multivariate time series," in *International Conference on Big Data*, 2018, pp. 1598–1605.

[27] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *AAAI Conference on Artificial Intelligence*, 2017, pp. 4278–4284.

[28] W. Chen and K. Shi, "Multi-scale attention convolutional neural network for time series classification," *Neural Networks*, vol. 136, pp. 126–140, 2021.

[29] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *International Journal of Computer Vision*, vol. 129, no. 6, pp. 1789–1819, 2021.

[30] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for thin deep nets," in *International Conference on Learning Representations*, 2015.

[31] J. Yim, D. Joo, J. Bae, and J. Kim, "A gift from knowledge distillation: Fast optimization, network minimization and transfer learning," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 7130–7138.

[32] D. Svitov and S. Alyamkin, "Margindistillation: Distillation for margin-based softmax," *arXiv preprint arXiv:2003.02586*, 2020.

[33] H. Oki, M. Abe, J. Miyao, and T. Kurita, "Triplet loss for knowledge distillation," in *International Joint Conference on Neural Networks*, 2020, pp. 1–7.

[34] J. H. Cho and B. Hariharan, "On the efficacy of knowledge distillation," in *International Conference on Computer Vision*, 2019, pp. 4793–4801.

[35] S. Mirzadeh, M. Farajtabar, A. Li, N. Levine, A. Matsukawa, and H. Ghasemzadeh, "Improved knowledge distillation via teacher assistant," in *AAAI Conference on Artificial Intelligence*, 2020, pp. 5191–5198.

[36] M. Friedman, "A comparison of alternative tests of significance for the problem of m rankings," *The Annals of Mathematical Statistics*, vol. 11, no. 1, pp. 86–92, 1940.

[37] 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.

[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] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *The Journal of Machine Learning Research*, vol. 7, pp. 1–30, 2006.

Xueyuan Gong is currently an assistant professor at the school of Intelligent Systems Science and Engineering, Jinan University, Zhuhai, China. He obtained his Ph.D. and M.Sc. degree in Computer Science from University of Macau in 2014 and 2019 respectively. Before that, he received his B.Sc. degree from Macau University of Science and Technology in 2011. His research interests include Machine Learning and Data Mining.

Yain-Whar Si is an associate professor at the University of Macau. He holds a Ph.D. degree in Information Technology from the Queensland University of Technology, Australia. His research interests are in the areas of Financial Technology, Computational Intelligence, and Information Visualization.

Yongqi Tian received his B.S. degree from Beijing Institute of Technology, China, in 2019. He is currently pursuing a master's degree in optical engineering at the Beijing Institute of Technology. His current research interests include Computer Vision and Deep Learning.

Cong Lin is currently an assistant professor at the School of Intelligent Systems Science and Engineering, Jinan University, Zhuhai, China. He obtained his Ph.D. degree in Software Engineering from University of Macau in 2017. His research interests include pattern recognition, machine learning, and computer vision. He is currently working on applicational image to image transforms using deep learning and vision-based object detectors for practical use.

Xinyuan Zhang received the B.S. and Ph.D. degrees in computer science from Sun Yat-sen University, Guangzhou, China, in 2014 and 2019, respectively. From 2015 to 2017, he was a Research Assistant with the Department of Electronic Engineering, City University of Hong Kong, Hong Kong, China. His current research interests include evolutionary computation algorithms, swarm intelligence algorithms, large-scale optimization their applications in real-world problems.

Xiaoxiang Liu is currently an associate professor at the School of Intelligent Systems Science and Engineering, Jinan University, Zhuhai, China. He obtained his Ph.D. degree, M.Sc. degree, and B.Sc. degree all in Aeronautical and Astronautical Manufacturing Engineering from Northwestern Polytechnical University. His research interests include Computer Vision and Artificial Intelligence.

TABLE III
THE ACCURACY OF DIFFERENT APPROACHES

Datasets	ROCKET	ITime	LSTime	KDTime	KDCT	KDCR
ACSF1	0.882 ± 0.01	0.896 ± 0.007	0.903 ± **0.004**	0.911 ± 0.021	**0.914** ± 0.006	0.913 ± 0.009
Adiac	0.8 ± **0.005**	0.822 ± 0.02	0.791 ± 0.007	0.842 ± 0.007	**0.847** ± 0.009	0.826 ± 0.014
ArrowHead	0.807 ± **0.009**	0.84 ± 0.018	0.854 ± 0.012	0.857 ± 0.016	0.859 ± 0.025	**0.873** ± 0.019
Beef	**0.8** ± **0.0**	0.767 ± **0.0**	0.767 ± **0.0**	0.793 ± 0.057	0.793 ± 0.025	**0.8** ± 0.047
BeetleFly	0.9 ± **0.0**	0.74 ± 0.02	0.8 ± 0.041	0.87 ± 0.024	**0.94** ± 0.02	0.83 ± 0.06
BirdChicken	0.89 ± **0.02**	0.92 ± 0.024	0.917 ± 0.085	**0.95** ± 0.055	0.92 ± 0.024	**0.95** ± 0.063
BME	**1.0** ± **0.0**	0.995 ± **0.0**	0.995 ± 0.004	0.999 ± 0.002	0.992 ± 0.003	**1.0** ± 0.002
Car	0.917 ± **0.0**	0.91 ± 0.017	0.867 ± 0.014	**0.937** ± 0.012	0.91 ± 0.023	0.917 ± 0.008
CBF	**1.0** ± **0.0**	0.998 ± 0.001	0.998 ± 0.001	0.999 ± **0.0**	0.999 ± **0.0**	**1.0** ± **0.0**
Chinatown	0.977 ± **0.0**	0.984 ± 0.005	**0.992** ± **0.0**	**0.992** ± 0.001	0.991 ± 0.002	**0.992** ± **0.0**
ChlorineConcentration	0.81 ± 0.006	0.839 ± 0.007	0.83 ± 0.016	0.864 ± **0.004**	0.848 ± 0.01	**0.873** ± 0.005
CinCECGTorso	0.826 ± **0.005**	0.863 ± 0.009	0.846 ± 0.007	0.868 ± 0.012	0.871 ± 0.009	**0.872** ± 0.008
Coffee	**1.0** ± **0.0**	**1.0** ± **0.0**	**1.0** ± **0.0**	**1.0** ± **0.0**	**1.0** ± **0.0**	**1.0** ± **0.0**
Computers	0.761 ± 0.005	0.813 ± 0.014	0.831 ± 0.005	0.826 ± **0.003**	**0.851** ± 0.014	0.818 ± 0.01
CricketX	0.829 ± 0.004	0.855 ± 0.014	0.852 ± 0.005	0.86 ± **0.002**	**0.871** ± 0.009	0.85 ± 0.008
CricketY	0.856 ± **0.002**	0.849 ± 0.017	0.873 ± 0.008	0.875 ± 0.006	0.879 ± 0.008	**0.893** ± 0.009
CricketZ	0.858 ± **0.003**	0.856 ± 0.007	0.858 ± **0.003**	**0.867** ± 0.006	0.865 ± 0.008	0.865 ± 0.007
Crop	0.763 ± **0.001**	0.77 ± 0.002	0.776 ± 0.003	0.777 ± 0.002	0.776 ± **0.001**	**0.788** ± **0.001**
DiatomSizeReduction	**0.975** ± **0.001**	0.93 ± 0.005	0.938 ± 0.023	0.965 ± 0.013	0.942 ± 0.013	0.951 ± 0.003
DistalPhalanxOutlineAgeGroup	0.758 ± **0.006**	0.795 ± 0.011	**0.845** ± 0.01	0.844 ± 0.012	0.836 ± 0.007	0.818 ± 0.01
DistalPhalanxOutlineCorrect	0.773 ± **0.005**	0.762 ± 0.017	0.815 ± 0.009	0.816 ± 0.011	**0.818** ± 0.011	0.808 ± 0.007
DistalPhalanxTW	**0.717** ± 0.007	0.626 ± 0.041	**0.717** ± **0.004**	0.635 ± 0.018	0.711 ± 0.016	0.713 ± 0.011
Earthquakes	**0.748** ± **0.0**	0.647 ± 0.038	0.724 ± 0.006	0.736 ± 0.013	0.742 ± 0.013	0.742 ± 0.012
ECG200	0.906 ± **0.005**	0.887 ± 0.013	0.905 ± 0.009	0.915 ± 0.01	0.921 ± **0.005**	**0.931** ± 0.007
ECG5000	**0.947** ± **0.0**	0.94 ± **0.0**	0.946 ± **0.0**	0.945 ± 0.001	**0.947** ± **0.0**	0.944 ± 0.001
ECGFiveDays	**1.0** ± **0.0**	0.998 ± 0.001	0.999 ± 0.001	**1.0** ± 0.001	0.999 ± 0.001	**1.0** ± **0.0**
ElectricDevices	0.73 ± **0.002**	0.697 ± 0.006	0.733 ± 0.008	0.741 ± 0.004	0.738 ± 0.006	**0.756** ± 0.004
EOGHorizontalSignal	0.578 ± **0.004**	0.661 ± 0.005	0.651 ± 0.007	0.691 ± 0.01	0.693 ± 0.005	**0.713** ± 0.007
EOGVerticalSignal	0.547 ± 0.006	0.496 ± 0.014	0.497 ± **0.005**	0.541 ± 0.017	0.542 ± 0.011	**0.563** ± 0.009
EthanolLevel	0.588 ± 0.008	0.783 ± 0.017	0.705 ± 0.017	0.762 ± **0.005**	**0.817** ± 0.01	0.816 ± 0.02
FaceAll	0.923 ± **0.007**	0.816 ± 0.01	0.833 ± 0.009	0.897 ± 0.028	0.912 ± 0.026	**0.963** ± 0.022
FaceFour	**0.975** ± 0.005	0.956 ± **0.0**	0.956 ± **0.0**	0.96 ± 0.008	0.954 ± 0.003	0.956 ± 0.008
FacesUCR	0.962 ± **0.001**	0.965 ± 0.002	0.964 ± 0.002	0.968 ± **0.001**	0.968 ± 0.002	**0.973** ± **0.001**
FiftyWords	**0.835** ± **0.001**	0.729 ± 0.003	0.63 ± 0.006	0.677 ± 0.022	0.753 ± 0.015	0.755 ± 0.014
Fish	0.983 ± **0.0**	0.984 ± 0.003	0.99 ± 0.004	**0.992** ± 0.004	0.99 ± 0.003	0.99 ± 0.003
FordA	0.944 ± 0.001	0.948 ± 0.002	0.958 ± **0.0**	**0.963** ± 0.001	0.961 ± 0.001	0.962 ± 0.001
FordB	0.803 ± 0.005	0.838 ± 0.003	0.856 ± 0.003	0.859 ± 0.004	0.861 ± **0.002**	**0.863** ± 0.004
FreezerRegularTrain	0.998 ± **0.0**	0.997 ± **0.0**	**0.999** ± **0.0**	**0.999** ± **0.0**	**0.999** ± **0.0**	0.998 ± **0.0**
FreezerSmallTrain	**0.952** ± **0.004**	0.872 ± **0.004**	0.874 ± 0.008	0.937 ± 0.006	0.935 ± 0.008	0.944 ± 0.007
Fungi	0.987 ± 0.003	0.999 ± **0.002**	0.995 ± 0.004	0.997 ± 0.003	0.997 ± 0.003	**1.0** ± 0.005
GunPoint	**1.0** ± **0.0**	**1.0** ± **0.0**	0.998 ± 0.002	0.999 ± 0.002	**1.0** ± **0.0**	**1.0** ± 0.006
GunPointAgeSpan	0.992 ± 0.003	0.985 ± 0.007	0.992 ± **0.001**	0.992 ± 0.005	0.992 ± 0.004	**0.994** ± 0.009
GunPointMaleVersusFemale	0.999 ± 0.002	**1.0** ± **0.0**	**1.0** ± **0.0**	**1.0** ± **0.0**	**1.0** ± **0.0**	**1.0** ± 0.002
GunPointOldVersusYoung	**1.0** ± **0.0**	**1.0** ± **0.0**	**1.0** ± **0.0**	**1.0** ± **0.0**	**1.0** ± **0.0**	**1.0** ± **0.0**
Ham	0.722 ± 0.02	0.767 ± 0.02	0.816 ± **0.009**	**0.821** ± 0.012	0.797 ± 0.011	0.818 ± 0.029
HandOutlines	0.944 ± **0.002**	0.94 ± 0.005	0.911 ± 0.011	0.945 ± 0.009	0.95 ± 0.006	**0.954** ± 0.01
Haptics	0.523 ± **0.0**	0.552 ± 0.014	0.576 ± 0.012	0.577 ± 0.006	**0.579** ± 0.013	0.562 ± 0.007
Herring	0.691 ± **0.012**	0.725 ± 0.035	**0.75** ± 0.013	0.728 ± 0.016	0.707 ± 0.023	0.743 ± 0.025
HouseTwenty	**0.965** ± **0.003**	0.934 ± **0.003**	0.948 ± 0.004	0.953 ± 0.012	0.943 ± 0.007	0.944 ± 0.016
InlineSkate	0.456 ± **0.004**	0.446 ± 0.015	0.418 ± 0.007	**0.474** ± 0.015	0.471 ± 0.031	0.458 ± 0.008
InsectEPGRegularTrain	**1.0** ± **0.0**	**1.0** ± **0.0**	**1.0** ± **0.0**	**1.0** ± **0.0**	**1.0** ± **0.0**	**1.0** ± **0.0**
InsectEPGSmallTrain	0.998 ± 0.002	**1.0** ± **0.0**	**1.0** ± **0.0**	**1.0** ± **0.0**	**1.0** ± **0.0**	**1.0** ± **0.0**
InsectWingbeatSound	**0.652** ± **0.001**	0.62 ± 0.009	0.635 ± 0.005	0.635 ± 0.007	0.631 ± 0.007	0.64 ± 0.004
ItalyPowerDemand	0.969 ± **0.0**	0.972 ± 0.002	0.973 ± 0.002	0.974 ± 0.001	0.974 ± 0.001	**0.975** ± 0.001
LargeKitchenAppliances	0.888 ± 0.003	0.89 ± 0.008	0.891 ± **0.002**	0.91 ± 0.004	0.897 ± 0.005	**0.914** ± 0.006
Lightning2	0.77 ± **0.0**	0.813 ± 0.017	0.831 ± 0.008	0.856 ± 0.019	0.859 ± 0.017	**0.869** ± 0.028
Lightning7	0.841 ± 0.011	0.822 ± 0.027	0.906 ± 0.011	0.905 ± **0.009**	0.916 ± 0.012	**0.938** ± 0.046
Mallat	0.955 ± **0.001**	0.95 ± 0.006	0.834 ± 0.011	0.947 ± 0.006	0.945 ± 0.007	**0.963** ± 0.014
Meat	0.95 ± **0.0**	0.913 ± 0.019	0.928 ± 0.031	0.913 ± 0.037	0.917 ± 0.028	**0.967** ± 0.012
MedicalImages	**0.802** ± **0.003**	0.761 ± 0.008	0.794 ± 0.007	0.786 ± **0.003**	0.79 ± 0.005	0.779 ± 0.004
MiddlePhalanxOutlineAgeGroup	0.603 ± **0.003**	0.562 ± 0.01	0.708 ± 0.004	0.582 ± **0.003**	**0.71** ± 0.006	0.706 ± 0.019
MiddlePhalanxOutlineCorrect	0.839 ± 0.007	0.824 ± 0.011	0.853 ± 0.004	**0.858** ± 0.007	0.855 ± **0.002**	0.854 ± 0.006
MiddlePhalanxTW	0.552 ± 0.006	0.498 ± 0.026	0.61 ± **0.002**	0.617 ± 0.006	**0.62** ± 0.009	0.598 ± 0.03
MixedShapesRegularTrain	0.971 ± **0.001**	0.975 ± 0.002	**0.976** ± **0.001**	**0.976** ± **0.001**	0.974 ± **0.001**	0.975 ± 0.002
MixedShapesSmallTrain	**0.937** ± **0.0**	0.932 ± 0.003	0.932 ± 0.004	**0.937** ± 0.001	0.935 ± 0.006	0.935 ± 0.006
MoteStrain	**0.91** ± **0.002**	0.9 ± 0.008	0.908 ± 0.008	0.896 ± 0.005	0.908 ± 0.004	0.874 ± 0.026
NonInvasiveFetalECGThorax1	0.954 ± 0.002	0.958 ± 0.003	0.952 ± 0.005	0.961 ± 0.003	**0.964** ± 0.003	**0.964** ± **0.001**
NonInvasiveFetalECGThorax2	**0.97** ± **0.001**	0.959 ± 0.004	0.96 ± 0.002	0.961 ± 0.003	0.961 ± **0.001**	0.961 ± 0.002
OliveOil	**0.907** ± **0.013**	0.753 ± 0.016	0.722 ± 0.079	0.633 ± 0.191	0.64 ± 0.196	0.767 ± 0.033
OSULeaf	0.939 ± **0.004**	0.97 ± 0.007	0.97 ± **0.004**	**0.981** ± **0.004**	0.978 ± 0.005	0.972 ± 0.006
PhalangesOutlinesCorrect	0.836 ± 0.003	0.831 ± 0.008	**0.855** ± **0.002**	**0.855** ± 0.008	0.848 ± 0.003	0.851 ± 0.003
Phoneme	0.281 ± **0.002**	0.339 ± 0.007	0.344 ± 0.006	**0.346** ± 0.004	**0.346** ± 0.003	0.331 ± 0.003
PigAirwayPressure	**0.834** ± 0.028	0.577 ± 0.018	0.367 ± 0.032	0.418 ± 0.034	0.592 ± 0.02	0.57 ± **0.008**
PigArtPressure	0.962 ± 0.004	0.995 ± 0.002	0.863 ± 0.006	**0.996** ± **0.0**	**0.996** ± **0.0**	0.992 ± 0.014
PigCVP	**0.928** ± **0.0**	0.77 ± 0.073	0.677 ± 0.057	0.77 ± 0.018	0.854 ± 0.022	0.866 ± 0.025
Plane	**1.0** ± **0.0**	**1.0** ± **0.0**	**1.0** ± **0.0**	**1.0** ± **0.0**	**1.0** ± **0.0**	**1.0** ± 0.008
PowerCons	0.979 ± 0.005	0.984 ± 0.006	0.997 ± 0.002	**1.0** ± **0.0**	0.994 ± 0.004	**1.0** ± **0.0**
ProximalPhalanxOutlineAgeGroup	0.858 ± **0.002**	0.842 ± 0.007	0.866 ± **0.002**	**0.872** ± **0.002**	0.869 ± 0.005	0.868 ± 0.003
ProximalPhalanxOutlineCorrect	0.897 ± 0.007	0.895 ± 0.011	0.925 ± 0.009	0.918 ± **0.006**	0.925 ± 0.008	**0.927** ± 0.007
ProximalPhalanxTW	0.814 ± **0.004**	0.778 ± 0.018	**0.863** ± 0.006	0.862 ± 0.006	0.861 ± 0.005	0.852 ± 0.016
RefrigerationDevices	0.533 ± 0.01	0.498 ± 0.009	0.556 ± 0.014	0.565 ± 0.008	**0.566** ± **0.007**	0.529 ± 0.01
Rock	0.68 ± **0.0**	0.744 ± 0.039	0.62 ± 0.028	0.704 ± 0.067	0.733 ± 0.037	**0.768** ± 0.029
ScreenType	0.477 ± 0.014	0.604 ± 0.008	0.633 ± **0.003**	0.631 ± 0.006	0.635 ± 0.007	**0.65** ± 0.011
SemgHandGenderCh2	**0.901** ± **0.005**	0.878 ± 0.009	0.878 ± **0.005**	0.885 ± **0.005**	0.891 ± 0.006	0.88 ± 0.006
SemgHandMovementCh2	**0.591** ± 0.006	0.446 ± 0.018	0.451 ± **0.002**	0.462 ± 0.012	0.469 ± 0.023	0.461 ± 0.03
SemgHandSubjectCh2	**0.843** ± 0.007	0.652 ± 0.025	0.669 ± 0.011	0.693 ± **0.006**	0.695 ± 0.016	0.688 ± 0.029
ShapeletSim	**1.0** ± **0.0**	0.712 ± 0.03	0.997 ± 0.002	0.967 ± 0.041	0.991 ± 0.007	**1.0** ± 0.002
ShapesAll	0.91 ± **0.001**	0.911 ± 0.003	0.895 ± 0.004	0.916 ± 0.002	0.919 ± 0.003	**0.922** ± 0.003
SmallKitchenAppliances	**0.82** ± 0.006	0.753 ± 0.007	0.814 ± **0.004**	0.817 ± 0.007	0.813 ± 0.008	0.803 ± 0.008
SmoothSubspace	0.979 ± 0.003	0.974 ± 0.006	0.99 ± 0.004	0.993 ± 0.004	0.991 ± **0.002**	**1.0** ± 0.008
SonyAIBORobotSurface1	0.918 ± **0.001**	0.887 ± 0.014	0.916 ± 0.026	0.902 ± 0.026	0.903 ± 0.028	**0.969** ± 0.027
SonyAIBORobotSurface2	0.918 ± **0.003**	**0.97** ± 0.006	0.962 ± 0.004	0.961 ± **0.003**	0.964 ± **0.003**	0.953 ± 0.012
StarLightCurves	**0.981** ± 0.001	0.977 ± 0.001	0.98 ± **0.0**	0.98 ± **0.0**	**0.981** ± **0.0**	**0.981** ± **0.0**
Strawberry	0.981 ± 0.002	0.982 ± 0.002	0.983 ± 0.002	**0.985** ± **0.001**	0.983 ± 0.002	0.981 ± 0.002
SwedishLeaf	0.964 ± **0.002**	0.968 ± 0.005	0.972 ± 0.004	0.974 ± 0.004	0.973 ± 0.005	**0.975** ± 0.003
Symbols	0.974 ± **0.001**	0.981 ± 0.003	0.976 ± 0.005	0.983 ± **0.001**	**0.984** ± 0.002	0.975 ± 0.004
SyntheticControl	0.999 ± 0.002	0.996 ± 0.001	0.998 ± 0.001	0.998 ± 0.001	0.997 ± **0.0**	**1.0** ± 0.002
ToeSegmentation1	0.968 ± **0.002**	0.973 ± 0.006	0.98 ± 0.003	0.97 ± 0.006	0.976 ± 0.004	**0.988** ± 0.012
ToeSegmentation2	0.942 ± 0.006	0.956 ± 0.005	0.962 ± **0.002**	0.967 ± 0.01	0.964 ± 0.006	**0.969** ± 0.004
Trace	**1.0** ± **0.0**	**1.0** ± **0.0**	**1.0** ± **0.0**	**1.0** ± **0.0**	**1.0** ± **0.0**	**1.0** ± **0.0**
TwoLeadECG	**0.999** ± **0.0**	0.997 ± 0.002	0.998 ± **0.0**	**0.999** ± **0.0**	0.998 ± 0.001	**0.999** ± **0.0**
TwoPatterns	**1.0** ± **0.0**	**1.0** ± **0.0**	**1.0** ± **0.0**	**1.0** ± **0.0**	**1.0** ± **0.0**	**1.0** ± **0.0**
UMD	0.986 ± **0.0**	**0.995** ± **0.0**	0.993 ± 0.002	**0.995** ± **0.0**	**0.995** ± **0.0**	**0.995** ± 0.003
UWaveGestureLibraryAll	**0.975** ± **0.001**	0.894 ± 0.002	0.898 ± 0.002	0.897 ± 0.002	0.897 ± 0.003	0.899 ± 0.002
UWaveGestureLibraryX	**0.85** ± **0.001**	0.805 ± 0.004	0.817 ± 0.002	0.816 ± 0.002	0.816 ± 0.003	0.819 ± 0.002
UWaveGestureLibraryY	**0.773** ± **0.002**	0.728 ± 0.005	0.749 ± 0.004	0.741 ± 0.003	0.737 ± 0.004	0.744 ± 0.005
UWaveGestureLibraryZ	**0.792** ± **0.002**	0.761 ± **0.002**	0.77 ± 0.006	0.767 ± 0.003	0.768 ± 0.003	0.77 ± 0.003
Wafer	0.999 ± **0.0**	0.998 ± **0.0**	**1.0** ± **0.0**	0.999 ± **0.0**	0.999 ± **0.0**	0.998 ± 0.001
Wine	0.804 ± **0.022**	0.774 ± 0.025	0.833 ± 0.079	0.811 ± 0.156	0.833 ± 0.059	**0.926** ± 0.03
WordSynonyms	**0.753** ± **0.002**	0.692 ± 0.01	0.644 ± 0.011	0.707 ± 0.005	0.699 ± 0.006	0.707 ± 0.008
Worms	0.743 ± **0.01**	0.754 ± 0.03	0.738 ± 0.011	0.81 ± 0.02	**0.822** ± 0.018	0.752 ± 0.04
WormsTwoClass	0.79 ± 0.015	0.808 ± 0.03	0.891 ± 0.011	0.884 ± **0.009**	0.9 ± **0.009**	**0.906** ± 0.029
Yoga	**0.912** ± 0.005	0.909 ± **0.004**	0.898 ± **0.004**	0.908 ± 0.008	0.909 ± 0.005	0.91 ± 0.006
Total accuracy wins	40	11	17	30	29	54
Total std wins	76	21	35	34	27	17