Information Sciences 613 (2022) 184–203

Contents lists available at ScienceDirect

Information Sciences

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

KDCTime: Knowledge distillation with calibration on InceptionTime for time-series classification

Xueyuan Gong [a], Yain-Whar Si [b], Yongqi Tian [c], Cong Lin [a], Xinyuan Zhang [a], Xiaoxiang Liu [a,*]

[a] School of Intelligent Systems Science and Engineering, Jinan University, Zhuhai, China
[b] Department of Computer and Information Science, University of Macau, Macau, China
[c] School of Optoelectronics, Beijing Institute of Technology, Beijing, China

A R T I C L E I N F O

Article history:
Received 2 May 2022
Received in revised form 9 August 2022
Accepted 13 August 2022
Available online 18 August 2022

Keywords:
Time-series classification
Deep neural networks
InceptionTime
Overfitting
Knowledge distillation

A B S T R A C T

Time-series classification approaches based on deep neural networks easily overfit UCR datasets, which is caused by the few-shot problem of those datasets. Therefore, to alleviate the overfitting phenomenon to further improve accuracy, we first propose label smoothing for InceptionTime (LSTime), which adopts the soft label information compared to only hard labels. Next, instead of manually adjusting soft labels by LSTime, knowledge distillation for InceptionTime (KDTime) is proposed to automatically generate soft labels by the teacher model while compressing the inference model. Finally, to rectify the incorrectly predicted soft labels from the teacher model, knowledge distillation with calibration for InceptionTime (KDCTime) is proposed, which contains two optional calibrating strategies, i.e., KDC by translating (KDCT) and KDC by reordering (KDCR). The experimental results show that the KDCTime accuracy is promising, while its inference time is orders of magnitude faster than state-of-the-art approaches.

1. Introduction

Time-series classification (TSC) is one of the most challenging tasks in data mining [13]. In recent years, with the remarkable success of deep neural networks (DNNs) in computer vision (CV) [21,19,41], many researchers have attempted 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 easily overfit datasets from the UCR archive [1]. Specifically, several experiments are conducted for 3 typical DNNs, i.e., fully convolutional networks (FCN) [44], residual networks (ResNet) [19,44], and InceptionTime [14] on the UCR datasets. Selecting InceptionTime as an example, only 23 datasets have a training and test accuracy gap less than 0.05, which means the gap on the other 62 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 remains stationary until the end.

After thoroughly analyzing the UCR datasets, we claim that the difference between datasets in CV and those in the 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 DNNs,

* Corresponding author.
E-mail addresses: xygong@jnu.edu.cn (X. Gong), fstasp@um.edu.mo (Y.-W. Si), 3220190462@bit.edu.cn (Y. Tian), conglin@jnu.edu.cn (C. Lin), Zhangxy@jnu.edu.cn (X. Zhang), tlxx@jnu.edu.cn (X. Liu).

https://doi.org/10.1016/j.ins.2022.08.057
0020-0255/© 2022 Elsevier Inc. All rights reserved.

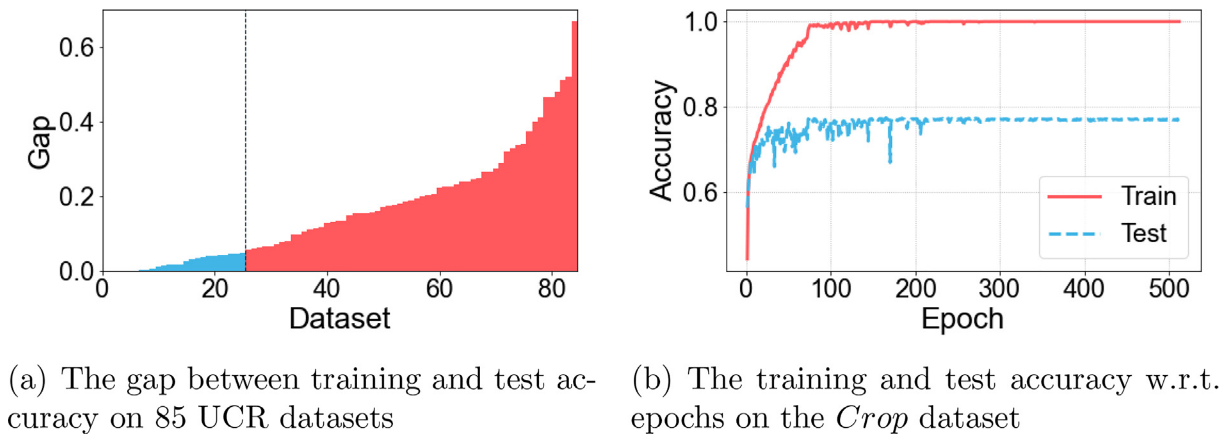


Fig. 1. Two examples showing that InceptionTime, including other DNNs, is easy to overfit on UCR datasets.

e.g., MNIST [27] and CIFAR10 [26] are both 5, 000-shot learning datasets with 50, 000 training samples in total, and ImageNet [12] is, on average, a 700-shot learning dataset with more than 14 million training samples. However, in the UCR archive, 58 datasets are less than 100-shot and 75 datasets with fewer than 1, 000 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 have adopted methods for alleviating overfitting without addressing the overfitting problem, such as the hierarchical vote system for a collective of transformation-based ensembles (HIVE-COTE) [30] with an ensemble of 37 classifiers, InceptionTime [14] with an ensemble of 5 models, and RandOm Convolutional KErnel Transform (ROCKET) [10] with *L*2 regularization and cross-validation. 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., *L*1 and *L*2 regularization, batch normalization (BN) [22], dropout [38], and early stopping, etc, we first proposed label smoothing based on InceptionTime [42] (LSTime) for improving the generalization ability of InceptionTime, as soft labels are closer to real life than hard labels. For instance, as shown in Fig. 2(a), the true label of that handwritten number is 2. However, it also resembles 3 intuitively. Thus, giving a hard label 2 to that number may cause information loss. Additionally, in stocks, there are many meaningful chart patterns, among which head-and-shoulders (H&S), triple-top (TT), and double-top (DT) [43] are similar. In Fig. 2(b), the H&S chart pattern is close to TT. Therefore, we wish to keep its TT information. As a consequence, soft labels maintain more information than hard labels. However, we would like to obtain the soft labels automatically rather than set them manually. Thus, knowledge distillation based on InceptionTime [20] (KDTime) is leveraged to generate soft labels by a teacher model, which is a pretrained network with a deep and complex architecture. The predicted labels from the teacher model represent the knowledge it learned. After that, the knowledge (predicted soft labels) can be distilled to help the student model training, which has a relatively smaller and simpler architecture. As a consequence, knowledge distillation can also

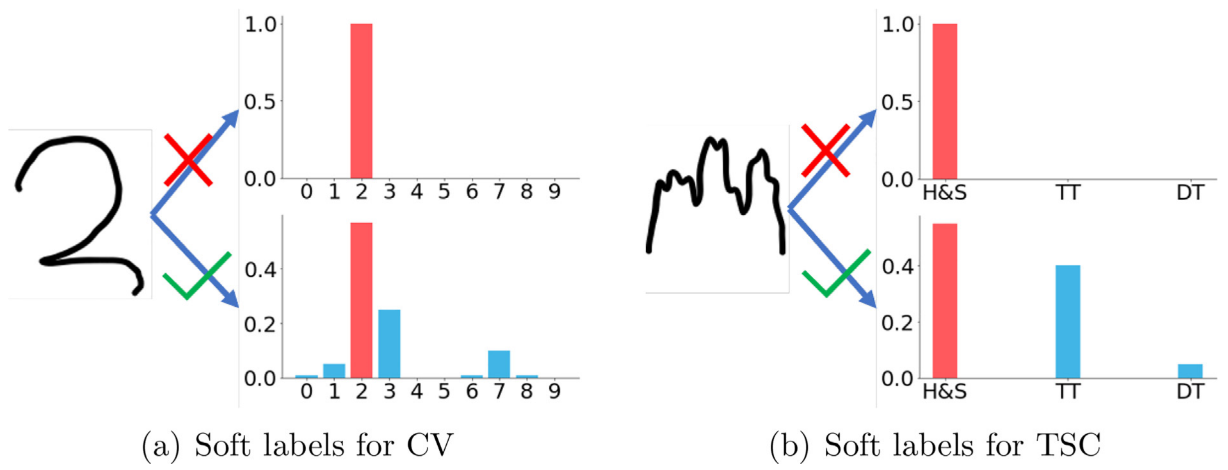


Fig. 2. Two examples demonstrating the benefits of soft labels.

reduce the inference time because of the student model. Finally, we claim that the teacher is not 100% correct, which means that it may produce incorrect soft labels and misguide the student. As the ground-truth labels have already been obtained, we propose simply calibrating the incorrectly predicted soft labels to maximize the InceptionTime accuracy, called knowledge distillation with calibration based on InceptionTime (KDCTime). In addition, KDCTime includes two optional calibrating strategies. KDC by translating (KDCT) and KDC by reordering (KDCR).

The InceptionTime model employed in LSTime, KDTime, and KDCTime is a single model instead of the 5 model since the training and inference times 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 the only one with 3 inception modules. In summary, the main contributions in this paper can be concluded as follows:

- discovered that the TSC approaches based on DNNs normally overfit the UCR datasets, which is caused by the few-shot problem of those datasets. Thus, the best direction to improve DNN performance is to alleviate overfitting.
- Combined label smoothing and knowledge distillation with InceptionTime denoted LSTime and KDTime, respectively. LSTime trains the InceptionTime model with manually controlled soft labels, while KDTime can 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: KDCT and KDCR. As a consequence, KDCTime further improves the KDTime accuracy.
- We have tested the accuracy, training time, and test time of Multiscale Attention CNN (MACNN)[7], echo memory-augmented network (EMAN)[31], ROCKET[10], InceptionTime[14], LSTime, KDTime, and KDCTime. The results show that compared to state-of-the-art methods, 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 2. Next, LSTime, KDTime, and KDCTime are introduced in Section 3. The experimental results are discussed in Section 4. Finally, Section 5 concludes the paper.

2. Related work

TSC, as a traditional time-series mining research direction, has been considered one of the most challenging problems [13]. Traditionally, 1 nearest neighbor (1-NN) classifiers based on dynamic time warping (DTW) distance have been shown to be a very promising approach [2]. However, the time complexity of DTW is unacceptable compared to the 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. [35] proposed UCR-DTW by leveraging lower bounding and early abandonment. Sakurai et al. [37] proposed SPRING under the DTW distance, which can monitor time-series streams in real time. Gong et al. [17] proposed forward-propagation NSPRING (FPNS) to further accelerate the speed of SPRING. Nevertheless, those studies all concentrated on time complexity. The upper bound of their accuracy is the accuracy of the DTW distance.

To break through the accuracy bottleneck, some researchers concentrate on DTW, e.g. Time-weight-based DTW [28] and Adaptive constrained DTW (ACDTW) [29], and some others focus on the representation of time-series, e.g. interpretable representation [4] and Shape-sphere [25]. In the meantime, researchers found that ensembling several classifiers can significantly improve the TSC accuracy. Thus, Baydogan et al. [5] selected an ensemble of decision trees. Kate [24] employed an ensemble of several 1-NN classifiers with different distance measures, including DTW distance. These methods motivated the development of an ensemble of 35 classifiers, named collective of transformation-based ensembles (COTE) [3], which ensembles those classifiers over different time-series representations instead of the same ones. After that, Lines et al. [30] 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, 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 [44]. Zheng et al. [47] proposed Denoising Temporal Convolutional Recurrent Autoencoder (DTCRAE) to combine a TCN encoder and a GRU decoder together. Pei et al. [34] proposed 3D Augmented Convolutional Network (3DACN) for multi-modal time-series data. Cui et al. [9] proposed multiscale convolutional neural networks (MCNNs), which transform the time series into several feature vectors and feed those vectors into a CNN model. Wang et al. [44] implemented several DNN models originating from CV, i.e., multilayer perceptrons (MLPs), fully convolutional networks (FCNs), and residual networks (ResNets) are used to test their performance in TSC, which provides a strong baseline for DNN-based approaches. Based on a more recent DNN structure, i.e., inception module [41], Karimi-Bidhendi et al. [23] proposed an approach to transform time-series into feature maps using a Gramian angular difference field (GADF) and fed those maps to an InceptionNet that is pretrained for image recognition. By extending a more recent version of InceptionNet, i.e., Inception-V4 [40], Fawaz et al. [14] proposed InceptionTime, which ensembles 5 Inception-based models to obtain promising accuracy. The multiscale attention convolutional neural network (MACNN) [7] adopted the attention mechanism to further improve MCNN accuracy. Instead of utilizing convolutions as parts of the model, RandOm Convolutional KErnel Transform (ROCKET) [10] 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 has the best accuracy and inference time balance, while other DNN-based methods always require a longer training and inference time. DNN-based methods always suffer from long training times, e.g., MACNN requires 3 days of running time for training 85 datasets from the UCR archive. This builds a considerable barrier for researchers to reimplement the approach.

We found that InceptionTime is a quite competitive approach. Using only the 1 InceptionTime model instead of ensembling 5 models, it has a slow 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 its accuracy by the information obtained from soft labels. The idea of utilizing soft labels was proposed by Szegedy et al. [42], which controls the smooth level of soft labels by manually setting a parameter ε. However, a better method for determining 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) [20]. Since then, many extensions of KD have been proposed [18]. Some studies [36,46] concentrated on letting the student model learn the feature maps, instead of soft labels, of the teacher model. In addition, Svitov et al. [39] leveraged the labels predicted 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 to save more execution time. Inspired by [8,32], the gap between the student model and the teacher model should be small, or the student model will have difficulty mimicking 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.

Finally, we propose a novel InceptionTime model based on knowledge distillation with calibration to calibrate the incorrectly predicted labels. After extensively searching related literature, one similar method called logit adjustment (LA) [45] was found. However, LA is not based on InceptionTime, and it focuses on images instead of time series. In addition, our approach is different from LA since it has two calibrating strategies with mathematical analysis. In conclusion, our approach is orthogonal to LA.

3. 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 3.1. Then, InceptionTime is briefly introduced in Section 3.2. Then, label smoothing for InceptionTime (LSTime) is demonstrated in Section 3.3. Next, knowledge distillation for InceptionTime (KDTime) is depicted in Section 3.4. Finally, knowledge distillation with calibration for InceptionTime (KDCTime) is illustrated in Section 3.5, where it contains 2 strategies: calibration by translating (CT) and calibration by ordering (CR).

3.1. Notations and definitions

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. Thus, 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. Note that $\sum_{i=1}^{C} y_i = 1$ always holds for any $\mathbf{y}$. To this point, the true label of $\mathbf{x}$ is defined as a 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. (1).

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

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 $\mathbf{Y}^h = \{\mathbf{y}_1^h, \mathbf{y}_2^h, \ldots, \mathbf{y}_M^h\}$, where each time series $\mathbf{x}_i$ corresponds to a true label $\mathbf{y}_i^h$. An InceptionTime model is treated as a function $\mathscr{F} \in \mathbb{F}$ 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 $\mathscr{F}$. The predicted label $\hat{\mathbf{y}}$ is produced by normalizing $\mathbf{z}$ with the softmax function (Eq. (2)). Thus, $\sum_{i=1}^{C} y_i = 1$ always holds.

$$\sigma(z_i) = \hat{y}_i = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}} \tag{2}$$

A loss function $\mathscr{L}$ is a function measuring the difference between the predicted label $\hat{\mathbf{y}}$ and the true label $\mathbf{y}^h$ to determine the performance of $\mathscr{F}$. Finally, we are in a position to define the problem, which is given as follows:

Definition 1. Given a dataset $\mathbf{D}$, find an $\mathscr{F}^{\star}$ minimizing the predefined $\mathscr{L}$. Formally, it is demonstrated in Eq. (3).

$$\mathscr{F}^{\star} = \underset{\mathscr{F} \in \mathbb{F}}{\operatorname{argmin}} \sum_{i=1}^{M} \mathscr{L}(\mathbf{y}_i^h, \sigma(\mathscr{F}(\mathbf{x}_i))) \tag{3}$$

In addition, important notations are also briefly summarized in Table 1.

3.2. InceptionTime

The ordinary softmax cross-entropy loss is adopted in InceptionTime [14]. We first implemented the one-model version of InceptionTime with 3 inception modules, denoted $\mathscr{F}_S$. Thus, the loss function of $\hat{\mathbf{y}} = \sigma(\mathscr{F}_S(\mathbf{x}))$ is given in Eq. (4).

$$\mathscr{L}_{CE}(\mathbf{y}^h, \hat{\mathbf{y}}) = -\sum_{i=1}^{C} y_i^h \log \hat{y}_i \tag{4}$$

where it is easy to know that the final loss $\mathscr{L}_{CE}$ is only related to y_c^h, as all the other y_i^h are 0 (Eq. (1)). In other words, only the result of $\log \hat{y}_c$ survives in the summation. Therefore, for simplicity, Eq. (4) can also be written as Eq. (5).

$$\mathscr{L}_{CE}(\mathbf{y}^h, \hat{\mathbf{y}}) = -\log \hat{y}_c \tag{5}$$

Note that Eq. (5) is the reason that the one-hot class label $\mathbf{y}$ is called the hard label, as it explicitly selects only the probability of $\mathbf{x}$ belonging to class c yet ignores 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 1, a soft version of $\mathbf{y}^h$ is more feasible in this case.

3.3. Label smoothing for InceptionTime

Second, following the assumption in Section 3.2, we implement softmax cross-entropy with label smoothing (LS) [42], with $\mathscr{F}_S$ as the model. Therefore, the equation of label smoothed $\mathbf{y}^h$ is given in Eq. (6), denoted $\mathbf{y}^l$.

$$y_i^l = \begin{cases} (1-\varepsilon) + \varepsilon/C, & \text{if } i = c \\ \varepsilon/C, & \text{if } i \neq c \end{cases} \tag{6}$$

where ε is the smoothing coefficient set by users, representing how much the label is smoothed. Note that after LS, y_i^l still satisfies $\sum_{i=1}^{C} y_i = 1$. Alternatively, Eq. (6) can also be written as Eq. (7).

$$y_i^l = (1-\varepsilon)y_i^h + \frac{\varepsilon}{C} \tag{7}$$

As a consequence, the softmax cross-entropy loss with LS is given in Eq. (8).

$$\begin{cases} \mathscr{L}_{LS}(\mathbf{y}^l, \hat{\mathbf{y}}) & = -\sum_{i=1}^{C} y_i^l \log \hat{y}_i \\ & = -\sum_{i=1}^{C} [(1-\varepsilon)y_i^h + \frac{\varepsilon}{C}] \log \hat{y}_i \\ & = -\sum_{i=1}^{C} (1-\varepsilon)y_i^h \log \hat{y}_i - \sum_{i=1}^{C} \frac{\varepsilon}{C} \log \hat{y}_i \\ & = (1-\varepsilon)\mathscr{L}_{CE}(\mathbf{y}^h, \hat{\mathbf{y}}) + \varepsilon(-\frac{1}{C}\sum_{i=1}^{C} \log \hat{y}_i) \end{cases} \tag{8}$$

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

Table 1
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 $\mathbf{D}$
$\mathscr{F}$	An InceptionTime model
$\mathbf{z}$	The output of $\mathscr{F}(\mathbf{x})$
σ	The softmax function
$\hat{\mathbf{y}}$	The predicted label of $\sigma(\mathbf{z})$
$\mathscr{L}$	A loss function

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

3.4. Knowledge distillation for InceptionTime

Third, we implemented knowledge distillation (KD) to help us generate soft labels in place of manually controlling them. Instead of manually setting up soft labels, Hinton et al. [20] proposed KD for generating soft labels by a teacher model, which has 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 it learned, denoted $\mathbf{y}^t$. Thus, $\mathbf{y}^t$ is an automatic soft label compared to the manual soft label $\mathbf{y}^l$. Note that the one-model version of InceptionTime with 6 inception modules is incorporated as the teacher model, denoted $\mathscr{F}_T$.

In addition, instead of directly using softmax cross-entropy loss, softmax with a temperature τ and Kullback–Leibler (KL) divergence loss are adopted. The softmax with τ equation is given in Eq. (9).

$$y_i^\tau = \frac{e^{z_i/\tau}}{\sum_{j=1}^{C} e^{z_j/\tau}} \tag{9}$$

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 that $\tau = 1$ denotes that the labels remain unchanged, while $\tau < 1$ or $\tau > 1$ represents that the labels are steeper or smoother, respectively. For an extreme example, if $\tau \to \infty$, we have $\forall i, y_i^\tau = 1/C$. Thus, 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. (10).

$$\mathscr{L}_{KL}(\mathbf{y}^{t_\tau}, \hat{\mathbf{y}}^\tau) = \sum_{i=1}^{C} y_i^{t_\tau} \log \frac{y_i^{t_\tau}}{\hat{y}_i^\tau} \tag{10}$$

After that, $\mathscr{L}_{KL}(\mathbf{y}^{t_\tau}, \hat{\mathbf{y}}^\tau)$, representing the soft label loss, and $\mathscr{L}_{CE}(\mathbf{y}^h, \hat{\mathbf{y}})$, representing the hard label loss, are incorporated into a whole to train the student model, which has a relatively small architecture with fewer parameters. This procedure is regarded as distilling the knowledge from a teacher model into a student model, called KD, to preserve the teacher model accuracy while reducing its time and space complexity. Note that $\mathscr{F}_S$ is selected as the student model, which is the one-model version of InceptionTime with 3 inception modules. The KD loss equation is given in Eq. (11).

$$\begin{cases} \mathscr{L}_{KD}(\mathbf{y}^h, \hat{\mathbf{y}}, \mathbf{y}^{t_\tau}, \hat{\mathbf{y}}^\tau) \\ = (1-\varepsilon)\mathscr{L}_{CE}(\mathbf{y}^h, \hat{\mathbf{y}}) + \varepsilon\tau^2 \mathscr{L}_{KL}(\mathbf{y}^{t_\tau}, \hat{\mathbf{y}}^\tau) \end{cases} \tag{11}$$

where ε controls the weight of $\mathscr{L}_{CE}(\mathbf{y}^h, \hat{\mathbf{y}})$ (Eq. (4)) and $\mathscr{L}_{KL}(\mathbf{y}^{t_\tau}, \hat{\mathbf{y}}^\tau)$ (Eq. (10)). Note that τ^2 is necessary because the scale of $\mathscr{L}_{KL}(\mathbf{y}^{t_\tau}, \hat{\mathbf{y}}^\tau)$ decreases after fine-tuning by τ. Thus, multiplying a τ^2 helps it to be the same scale as $\mathscr{L}_{CE}(\mathbf{y}^h, \hat{\mathbf{y}})$, so that the total loss has no preference for $\mathscr{L}_{CE}(\mathbf{y}^h, \hat{\mathbf{y}})$.

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. Specifically, the incorrect soft labels generated by the teacher model will also result in incorrect labels predicted by the student model. To alleviate the effect of incorrect labels, KD adopts $\mathscr{L}_{CE}(\mathbf{y}^h, \hat{\mathbf{y}})$ and τ. Nonetheless, that brings additional hyperparameters into the model. Therefore, we propose a better method to alleviate the effect of incorrect labels while not bringing additional hyperparameters.

3.5. Knowledge distillation with calibration for InceptionTime

Finally, we propose knowledge distillation with calibration (KDC) to calibrate the incorrect soft labels generated by the teacher model before distillation. Note that the teacher model and student model are $\mathscr{F}_T$ and $\mathscr{F}_S$, respectively (Section 3.4). In this way, it is not necessary to employ $\mathscr{L}_{CE}(\mathbf{y}, \hat{\mathbf{y}})$ and τ in KDC. 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 $\sum_{i=1}^{C} y_i = 1$, 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 the figures. Note that distinct colors represent the areas of distinct classes. Thus, it is possible 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 (b).

Therefore, our main task is to propose a proper method to modify $\mathbf{y}^t$ to its correct area while its new position is located between $\mathbf{y}^t$ and $\mathbf{y}^h$. The calibrated $\mathbf{y}^t$ is denoted as $\mathbf{y}^{t_\zeta}$. To this end, two approaches for calibration are proposed: calibration by

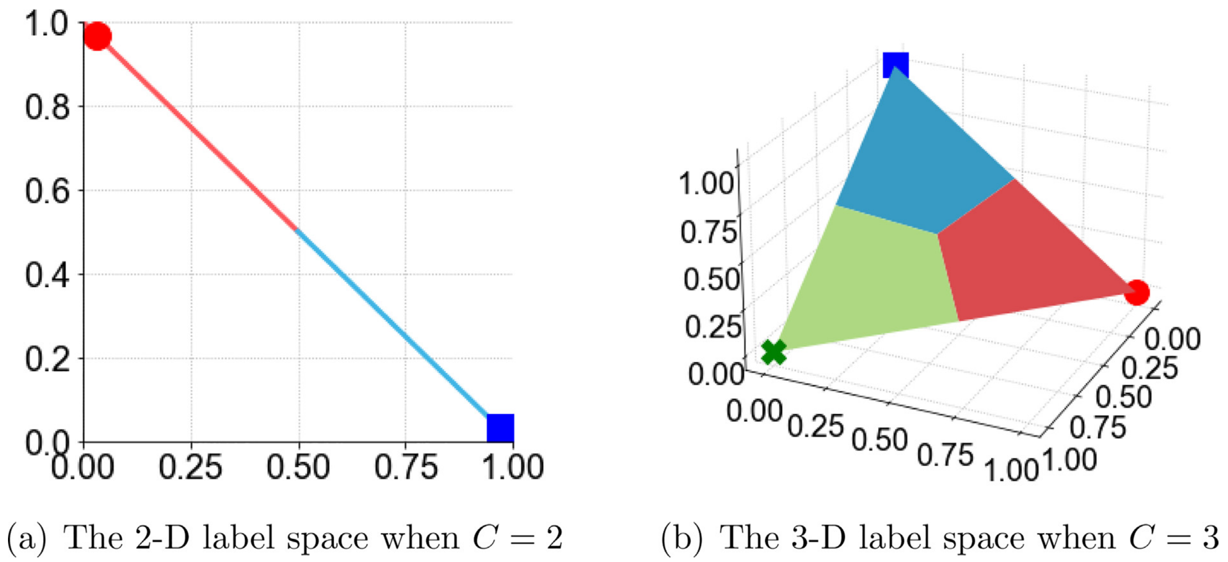


Fig. 3. Two examples showing the label spaces when $C = 2$ and $C = 3$.

translating and calibration by reordering. Note that only incorrectly predicted labels will be calibrated. Formally, given a $\mathbf{y}^t$ and its corresponding $\mathbf{y}^h, \mathbf{y}^{t_\zeta}$ will be computed only when $\arg\max_i\{y_i^t\} \neq \arg\max_i\{y_i^h\}$. In other words, $\arg\max_i\{y_i^t\} \neq c$.

3.5.1. Calibration by translating

Calibration by translating (CT) geometrically translates $\mathbf{y}^t$ from its original position to $\mathbf{y}^h$, the equation of which is shown in Eq. (12).

$$\mathbf{y}^{t_\zeta} = \mathbf{y}^t + \omega(\mathbf{y}^h - \mathbf{y}^t) \tag{12}$$

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 toward $\mathbf{y}^h$. It is easy to know that $\mathbf{y}^{t_\zeta} = \mathbf{y}^h$ when $\omega = 1$, and $\mathbf{y}^{t_\zeta} = \mathbf{y}^t$ 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. (13).

$$\omega = \frac{\delta}{\|\mathbf{y}^h - \mathbf{y}^t\|_2} \tag{13}$$

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 $\|\mathbf{y}^h - \mathbf{y}^t\|_2 \geqslant \delta$ always holds.

To this end, we calculated δ and obtained the magic number $\delta = 1/\sqrt{2}$. The calculation procedure is given in Appendix A. As a consequence, Eq. (12) can also be rewritten as (14).

$$\mathbf{y}^{t_\zeta} = \mathbf{y}^t + \frac{1}{\sqrt{2}} \frac{\mathbf{y}^h - \mathbf{y}^t}{\|\mathbf{y}^h - \mathbf{y}^t\|_2} \tag{14}$$

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

3.5.2. Calibration by reordering

Calibration by reordering (CR) represents reprioritizing the values of the incorrectly predicted label based on a specific strategy. Specifically, 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.

Given a $\mathbf{y}^t$ awaiting reordering, the reordering strategy is designed as follows. 1) $\mathbf{y}^t$ is sorted in descending order. The sorted $\mathbf{y}^t$ is denoted $\mathbf{y}_\mathbf{s}^\mathbf{t}$. Thus, since the descending order of y_i^t is random, we have $\mathbf{y}_\mathbf{s}^\mathbf{t} = \{y_1^{t_s}, y_2^{t_s}, \ldots, y_C^{t_s}\} = \{y_{1st}^t, y_{2nd}^t, \ldots y_{Cth}^t\}$, where y_{1st}^t is the largest y_i^t, y_{2nd}^t is the second largest y_i^t, and so on. Note that $\mathbf{y}_\mathbf{s}^\mathbf{t}$ is in descending order, which means $y_1^{t_s} = y_{1st}^t = \max_i\{y_i^t\}$. 2) After defining a temporary value $y_{tmp}^t = y_{1st}^t$, the value of $y_{(i+1)th}^t$ will be assigned to y_{ith}^t, from y_{1st}^t, y_{2nd}^t, all the way to $y_{ith}^t = y_c^t$. 3) Assign the value of y_{tmp}^t to y_c^t.

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

Algorithm 1 Algorithm of calibration by reordering

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

3.5.3. Analysis of CT and CR

We theoretically analyzed the difference between CT and CR in terms of calibration. Specifically, for one $\mathbf{y}^t$ that is not located in the correct area in the label space, KDC is utilized to calibrate $\mathbf{y}^t$ as $\mathbf{y}^{t_\zeta}$ either by CT or CR, where $\mathbf{y}^{t_\zeta}$ 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 $\mathscr{D}(\mathbf{y}^h, \mathbf{y}^t) = \|\mathbf{y}^h - \mathbf{y}^t\|_2$. As $\mathbf{y}^t$ in the red area (correct area) does not require calibration, their distance is calculated by $\mathscr{D}(\mathbf{y}^h, \mathbf{y}^t)$ directly. In contrast, $\mathbf{y}^t$ outside the red area (incorrect area) are calibrated as $\mathbf{y}^{t_\zeta}$. Therefore, their distance is calculated by $\mathscr{D}(\mathbf{y}^h, \mathbf{y}^{t_\zeta})$. As shown in Fig. 4(a), $\mathbf{y}^t$ close to the edge of the red area are mapped close to the vertex, i.e., the hard label $\mathbf{y}^h$, while $\mathbf{y}^t$ far away from the red area are mapped to the edge of the red area. However, in Fig. 4(b), the $\mathbf{y}^t$ close to the edge of the red area are mapped close to the edge, while the $\mathbf{y}^t$ far away from the red area are mapped close to $\mathbf{y}^h$. As a consequence, CT maintains 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 a 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. 15.

$$\mathscr{L}_{KDC}(\mathbf{y}^{t_\zeta}, \hat{\mathbf{y}}) = \mathscr{L}_{KL}(\mathbf{y}^{t_\zeta}, \hat{\mathbf{y}}) = \sum_{i=1}^{C} y_i^{t_\zeta} \log \frac{y_i^{t_\zeta}}{\hat{y}_i} \tag{15}$$

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

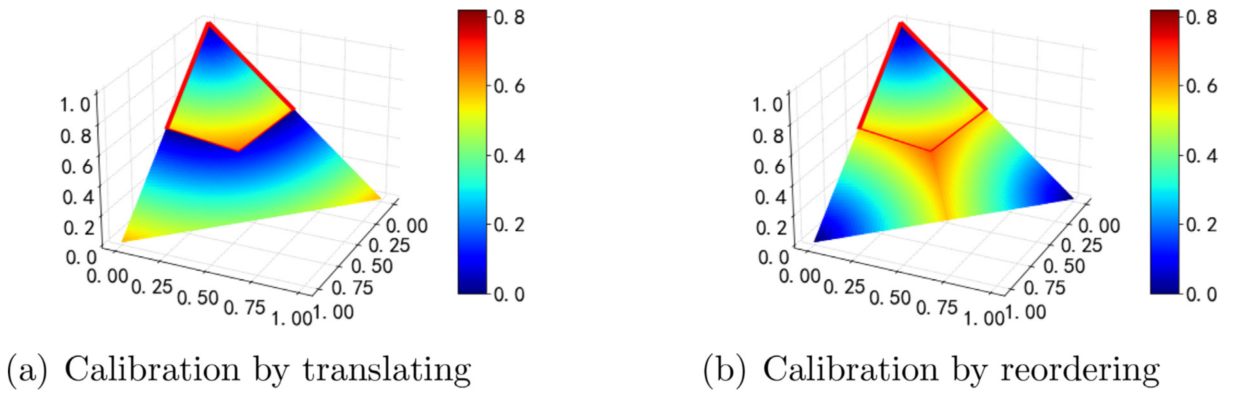


Fig. 4. A 3-D example illustrating the color map of the distance between $\mathbf{y}^h$ and $\mathbf{y}^t$ after KDC in the label space.

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 KDC algorithm is given in Algorithm 2.

Algorithm 2:Algorithm of KDCTime

```
Input: The training data $\mathbf{X}$ and its corresponding labels $\mathbf{Y}$.
Output: The trained student model $\mathscr{F}_S^*$
1: Initialize a teacher model $\mathscr{F}_T$
2: Train $\mathscr{F}_T$ by $\mathbf{X}$ and $\mathbf{Y}$ to obtain $\mathscr{F}_T^*$
3: Generate $\mathbf{Y}^t$ by $\mathscr{F}_T^*$
4: for each $\mathbf{y}_i^t$ do
5:     if $\mathbf{y}_i^t$ and $\mathbf{y}_i^h$ belong to distinct classes then
6:         Calibrate $\mathbf{y}_i^t$ to obtain $\mathbf{y}_i^{t_\zeta}$          ▹Eq. 14 or Algorithm 1
7: Initialize the student model $\mathscr{F}_S$
8: Train $\mathscr{F}_S$ by $\mathbf{X}$ and $\mathbf{Y}^{t_\zeta}$ to obtain $\mathscr{F}_S^*$
```

To better illustrate the KDC algorithm, we also present the framework and the flowchart corresponding to the KDC algorithm in Fig. 5 and Fig. 6. As shown in Fig. 5, a trained teacher model must be prepared first. After that, training data are fed into the teacher model to obtain soft labels and calibrate the incorrect labels if necessary. Finally, the student model is trained based on the calibrated soft labels and the hard labels. This process is illustrated in Fig. 6 from the perspective of flowcharts.

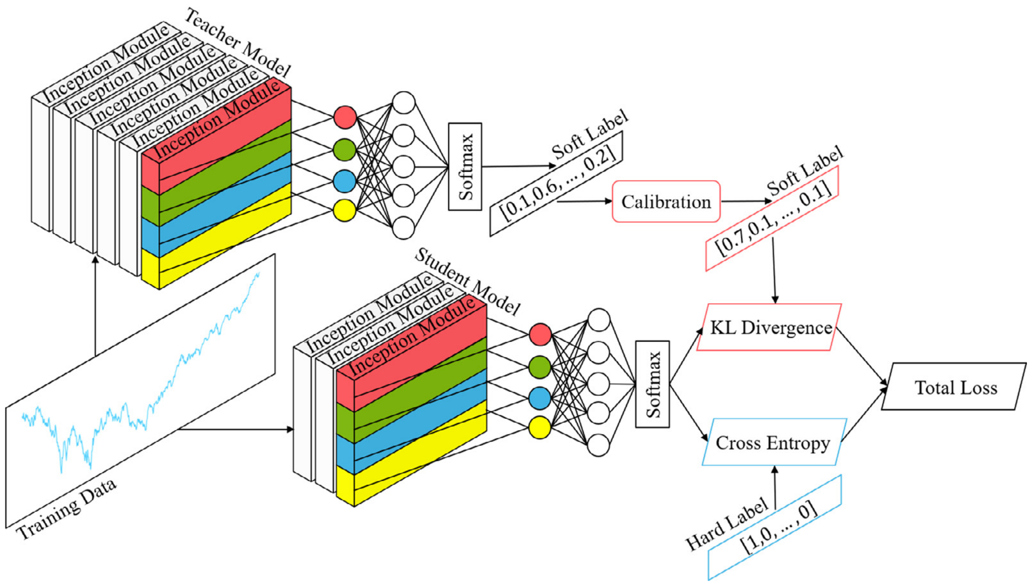


Fig. 5. The framework corresponding to the KDC algorithm.

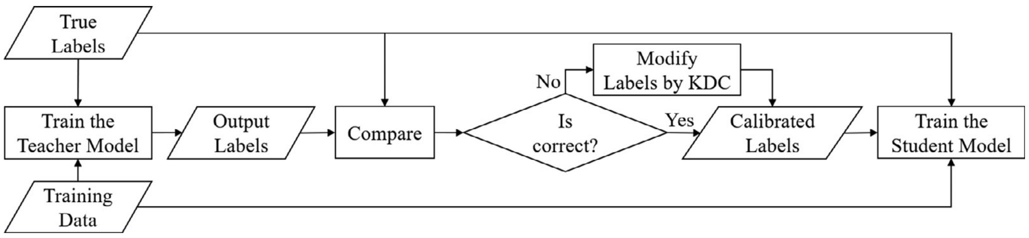


Fig. 6. The flowchart corresponding to the KDC algorithm.

3.5.4. Analysis of the student model and teacher model

To deeply demonstrate the efficiency of the student model, we analyzed the parameters (Params) and FLOPs (FLoating-point of OPerations) of the student model and the teacher model. Therefore, the compression ratio of space and time can be clearly shown with these two indices. Briefly, the main contributors in the two models are the inception module and the fully connected layer. Note that the final fully connected layer will be slightly different with a distinct number of classes, and thus, we select 10 classes as the example since it is close to the average of the class number in the UCR dataset.

Compared to the teacher model with 6 inception layers, Params of the student model with 3 inception layers are compressed from 66.4 K to 26.3 K. Overall, the compression ratio of the student model in space is nearly 3 times. Similar to Params, the FLOPs of the student model are compressed from 340.3 M to 134.2 M, which is also approximately a 3 times compression ratio. Therefore, the student model selected in this paper is efficient, where the specific experimental results corresponding to training and test time are shown in Section 4.5.

4. Experiments

Following the experimental settings of MACNN [7], EMAN [31], and ROCKET [10], we conduct the experiments on UCR datasets. Overall, MACNN, EMAN, 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. KDC by translating (KDCT) and KDC by reordering (KDCR). Except for MACNN, EMAN, and ROCKET, the other aforementioned methods can be concluded to be ITime-based approaches.

In the experiments, we first introduced the experimental setup. datasets and implementation details. Next, to find the best hyperparameters for LSTime, KDTime, KDCT, and KDCR, we conducted hyperparameter experiments. Note that ITime does not have any hyperparameter to be tuned. After that, we tested the accuracy, training time, and test time of all approaches. In addition, we visualized the features produced by the student and teacher models through t-SNE. Finally, to demonstrate the capability of alleviating overfitting for KDC, the accuracy gap between the training set and test set for ITime (without KDC) and KDCTime (with KDC) are compared.

Similar to [14], critical difference diagrams are drawn in the paper to better illustrate the results of different approaches, as the results of the 85 datasets are difficult to depict clearly. The critical difference diagram is a diagram drawn by the following steps. 1) Execute the Friedman test [15] to reject the null hypothesis. 2) Perform the pairwise post hoc analysis [6] by a Wilcoxon signed-rank test with Holm's alpha (5%) correction [16]. 3) Visualize the statistical result by [11], where a thick horizontal line represents that the approaches are not significantly different with respect to the results.

4.1. Dataset

UCR datasets contain 85 datasets with various lengths, numbers of classes, and numbers of training and test samples. They are always selected as the datasets for testing the performance of time-series classification approaches, including traditional and deep learning-based approaches. Thus, we train the proposed KDCTime, including KDCT and KDCR, on UCR datasets and compare it with other state-of-the-art approaches to show the performance of KDCTime.

4.2. Implementation details

We implement KDCTime with a student model containing 3 inception modules and a teacher model containing 6 ones. The teacher model is trained first, and then it guides the training process of the student model. These two models are trained for 512 epochs with batch size 64. The learning rate is set to 0.01 with a decay factor of 0.5 for every 35 epoch. ϵ is set to 0.5 to balance the contribution of soft labels and hard labels. Note that the proposed KDCTime does not need the temperature τ, which is commonly required in other knowledge distillation methods. In addition, they are both trained using the Adam optimizer with $\beta_1 = 0.9$ and $\beta_2 = 0.99$ on a single GPU. All hyperparameters are selected based on the study in Section 4.3. At testing time, only the student model remained to perform inference.

Our experiments are conducted on a computer equipped with an Intel Core i9-11900 CPU at 2.50 GHz, 32 GB 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.

4.3. 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 also be determined 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.

4.3.1. Batch size

First, the batch size was set to 64 without searching. The reason is that theoretically, a larger batch size is better. An extreme case is the full-batch. However, in deep learning tasks, this always leads to a consequence where graphics memory on GPU is infeasible to load the large dataset. Additionally, the batch size and the epoch depend on each other. A larger batch size represents more epochs for convergence, 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.

4.3.2. Epoch

With a fixed batch size 64, we compare the accuracy of 5 different epochs, which are $64, 128, 256, 512$, and 1024. The critical difference diagram is given in Fig. 7, where 1024 epochs have the best accuracy. However, 1024 epochs are of no critical difference from 512 epochs. Since the training time of DNN-based approaches is slow, 512 is selected as the epoch to save the training time. In addition, we also employ the early stop strategy with patience equals to 80 epochs for reducing the training time and alleviating overfitting.

4.3.3. Learning rate

The accuracy of 5 distinct learning rates are tested in total, which are $0.1, 0.01, 0.001, 0.0001$, and 0.00001. Moreover, learning rate decay was employed 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 is multiplied by 0.5 for every 70 epochs. Note there are 512 epochs, which ensures 7 times of decay in total. The critical difference diagram is shown in Fig. 8, where the learning rate equal to 0.01 has the highest accuracy. Thus, 0.01 is employed as the learning rate in the paper.

4.3.4. ε in LSTime

The smoothing coefficient ε (Eq. (8)) represents the smoothed level of labels. We test the accuracy of 5 different ε in LSTime, which are $0.1, 0.3, 0.5, 0.7$, and 0.9. The critical difference diagram is given in Fig. 9, where 0.5 obtains the best accuracy. This claims that ε should not be too small or too large, since a small ε gives the label little additional information, while a large ε causes too much information loss from its original class. In addition, the accuracy of large ε is less than that of small ε, which means that information from its original class is important, and it is not a good idea to completely abandon that information.

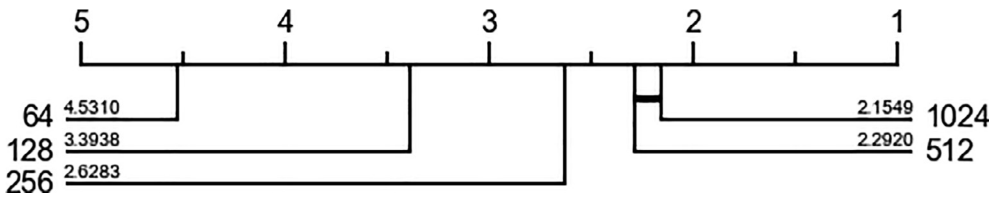


Fig. 7. Critical difference diagram for different epochs.

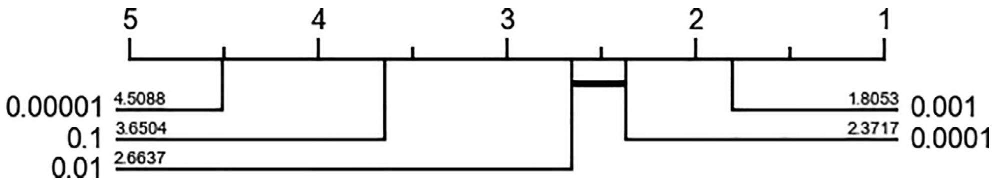


Fig. 8. Critical difference diagram for different learning rates.

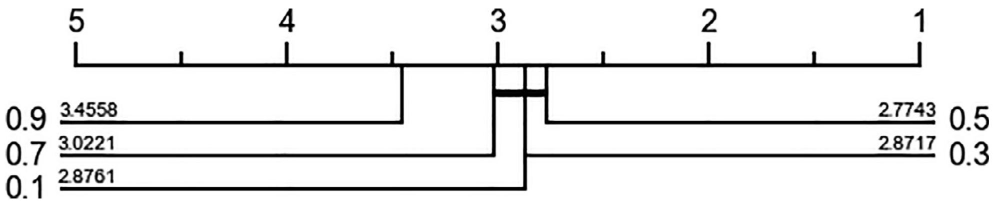


Fig. 9. Critical difference diagram for different ε in LSTime.

4.3.5. ε and τ in KDTime

KDTime contains 2 hyperparameters, which are ε and τ (Eq. (11)), where ε controls the weight of losses between the hard label and the smooth label, and τ fine-tunes the smoothed level of smooth labels (labels predicted by the 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. 10 and Fig. 11. Fig. 10 shows that $\varepsilon = 0.5$ has the best accuracy. The ε close to 0 or 1 will reduce the accuracy. In addition, Fig. 11 shows that $\tau = 8$ is the best. Similarly, the accuracy of KDTime will decrease if τ is too large or too small.

Thus, we 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, and ε and τ in KDTime are set to 0.5 and 8, respectively. Finally, Adam is adopted as the optimizing algorithm to update the model.

4.4. Accuracy of different approaches

In this section, we compare the accuracy of MACNN, EMAN, ROCKET, ITime, LSTime, KDTime, KDCTime by translating (KDCT), and KDCTime by reordering (KDCR). The accuracy of all approaches is averaged from 10 times running. The results are listed in Table 2 (Appendix B). In addition, the bold number represents the best accuracy among all approaches.

As shown in Table 2, KDCR obtains the highest accuracy on the 30 dataset, which indicates that its accuracy is competitive and promising. In addition, the critical difference diagram is also given to better illustrate the result of Table 2, which is shown in Fig. 12. Note that Fig. 12 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. In addition, KDCR and KDCT are not significantly different from KDTime. The reason is that the accuracy of 66 datasets is more than 0.8 for the teacher model, which claims that the majority of datasets meet 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 66 datasets dominate the other 19 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. 13.

In summary, KDCR has 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 labels located in the middle area of the label space. In Fig. 4(b), we know that 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 the other classes. However, KDCT calibrates the marginal labels close to $\mathbf{y}^h$, which loses that information. In addition, KDCR calibrates the labels close to the other classes as close to the correct class, as shown in Fig. 4(b), which eliminates the misguiding from deterministic incorrect labels. Nevertheless, KDCT maintains the information of those labels from incorrect classes.

4.5. Training and test times of different approaches

In this section, we compared the training and test times of MACNN, EMAN, ROCKET, ITime, LSTime, KDTime, and KDCTime, ignoring KDCT or KDCR, as their training and inference times were not significantly different. Instead of showing all the results in a table, diagrams of comparison were selected to better illustrate the results.

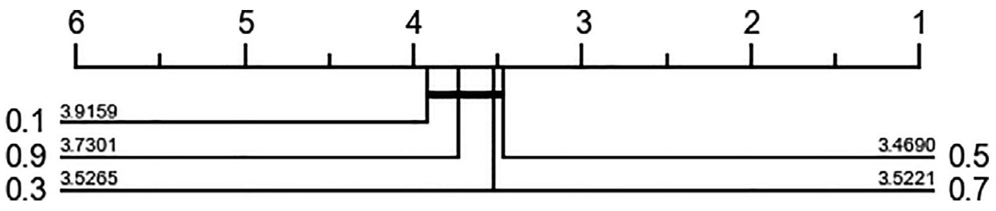


Fig. 10. Critical difference diagram for different ε in KDTime.

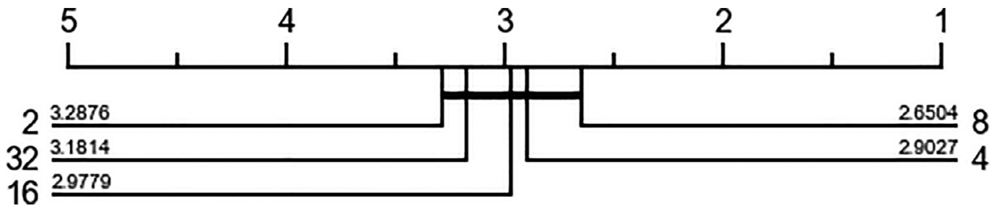


Fig. 11. Critical difference diagram for different τ in KDTime.

Table 2
The accuracy of different approaches.

Datasets	EMAN	MACNN	ROCKET	ITime	LSTime	KDTime	KDCT	KDCR
Adiac	0.808	0.812	0.8	0.822	0.791	0.842	**0.847**	0.826
ArrowHead	0.846	0.855	0.807	0.84	0.854	0.857	0.859	**0.873**
Beef	0.7	**0.907**	0.8	0.767	0.767	0.793	0.793	0.8
BeetleFly	0.85	**0.995**	0.9	0.74	0.8	0.87	0.94	0.83
BirdChicken	0.9	**0.99**	0.89	0.92	0.917	0.95	0.92	0.95
Car	0.883	0.91	0.917	0.91	0.867	**0.937**	0.91	0.917
CBF	**1.0**	0.999	**1.0**	0.998	0.998	0.999	0.999	**1.0**
ChlorineConcentration	0.796	**0.882**	0.81	0.839	0.83	0.864	0.848	0.873
CinCECGTorso	0.664	**0.877**	0.826	0.863	0.846	0.868	0.871	0.872
Coffee	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**
Computers	0.756	0.814	0.761	0.813	0.831	0.826	**0.851**	0.818
CricketX	0.805	0.854	0.829	0.855	0.852	0.86	**0.871**	0.85
CricketY	0.785	0.863	0.856	0.849	0.873	0.875	0.879	**0.893**
CricketZ	0.81	**0.87**	0.858	0.856	0.858	0.867	0.865	0.865
DiatomSizeReduction	0.948	0.97	**0.975**	0.93	0.938	0.965	0.942	0.951
DistalPhalanxOutlineAgeGroup	**0.853**	0.762	0.758	0.795	0.845	0.844	0.836	0.818
DistalPhalanxOutlineCorrect	**0.828**	0.775	0.773	0.762	0.815	0.816	0.818	0.808
DistalPhalanxTW	**0.8**	0.682	0.717	0.626	0.717	0.635	0.711	0.713
Earthquakes	**0.814**	0.748	0.748	0.647	0.724	0.736	0.742	0.742
ECG200	0.92	0.915	0.906	0.887	0.905	0.915	0.921	**0.931**
ECG5000	0.945	0.946	**0.947**	0.94	0.946	0.945	**0.947**	0.944
ECGFiveDays	**1.0**	0.997	**1.0**	0.998	0.999	0.999	**1.0**	**1.0**
ElectricDevices	0.717	0.691	0.73	0.697	0.733	0.741	0.738	**0.756**
FaceAll	0.904	0.869	0.923	0.816	0.833	0.897	0.912	**0.963**
FaceFour	0.943	0.956	**0.975**	0.956	0.956	0.96	0.954	0.956
FacesUCR	0.945	**0.976**	0.962	0.965	0.964	0.968	0.968	0.973
FiftyWords	0.756	**0.862**	0.835	0.729	0.63	0.677	0.753	0.755
Fish	0.937	0.989	0.983	0.984	0.99	0.99	0.99	**0.992**
FordA	0.928	0.95	0.944	0.948	0.958	0.961	**0.963**	0.962
FordB	**0.914**	0.868	0.803	0.838	0.856	0.859	0.861	0.863
GunPoint	0.993	0.998	**1.0**	**1.0**	0.998	0.999	**1.0**	**1.0**
Ham	0.762	**0.822**	0.722	0.767	0.816	0.821	0.797	0.818
HandOutlines	0.884	0.938	0.944	0.94	0.911	0.945	0.95	**0.954**
Haptics	0.494	0.532	0.523	0.552	0.576	0.577	**0.579**	0.562
Herring	0.625	0.675	0.691	0.725	**0.75**	0.728	0.707	0.743
InlineSkate	0.438	**0.483**	0.456	0.446	0.418	0.474	0.471	0.458
InsectWingbeatSound	0.643	0.643	**0.652**	0.62	0.635	0.635	0.631	0.64
ItalyPowerDemand	0.966	0.968	0.969	0.972	0.973	0.974	0.974	**0.975**
LargeKitchenAppliances	0.904	**0.915**	0.888	0.89	0.891	0.91	0.897	0.914
Lightning2	0.82	0.803	0.77	0.813	0.831	0.856	0.859	**0.869**
Lightning7	0.863	0.841	0.841	0.822	0.906	0.905	0.916	**0.938**
Mallat	0.958	**0.975**	0.955	0.95	0.834	0.947	0.945	0.963
Meat	0.95	**0.978**	0.95	0.913	0.928	0.913	0.917	0.967
MedicalImages	0.757	0.774	**0.802**	0.761	0.794	0.786	0.79	0.779
MiddlePhalanxOutlineAgeGroup	**0.8**	0.609	0.603	0.562	0.708	0.582	0.71	0.706
MiddlePhalanxOutlineCorrect	0.815	0.829	0.839	0.824	0.853	**0.858**	0.855	0.854
MiddlePhalanxTW	**0.647**	0.581	0.552	0.498	0.61	0.617	0.62	0.598
MoteStrain	0.889	**0.91**	**0.91**	0.9	0.908	0.896	0.908	0.874
NonInvasiveFetalECGThorax1	0.939	0.945	0.954	0.958	0.952	0.961	**0.964**	**0.964**
NonInvasiveFetalECGThorax2	0.933	0.953	**0.97**	0.959	0.96	0.961	0.961	0.961
OliveOil	0.833	0.85	**0.907**	0.753	0.722	0.633	0.64	0.767
OSULeaf	0.905	0.971	0.939	0.97	0.97	**0.981**	0.978	0.972
PhalangesOutlinesCorrect	0.831	0.826	0.836	0.831	**0.855**	0.848	**0.855**	0.851
Phoneme	0.234	0.336	0.281	0.339	0.344	**0.346**	**0.346**	0.331
Plane	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**
ProximalPhalanxOutlineAgeGroup	0.868	0.847	0.858	0.842	0.866	0.868	0.869	**0.872**
ProximalPhalanxOutlineCorrect	0.887	0.916	0.897	0.895	0.925	0.918	0.925	**0.927**
ProximalPhalanxTW	0.83	0.787	0.814	0.778	**0.863**	0.862	0.861	0.852
RefrigerationDevices	0.552	**0.579**	0.533	0.498	0.556	0.565	0.566	0.529
ScreenType	0.536	0.635	0.477	0.604	0.633	0.631	0.635	**0.65**
ShapeletSim	0.994	0.994	**1.0**	0.712	0.997	0.967	0.991	**1.0**
ShapesAll	0.878	**0.933**	0.91	0.911	0.895	0.916	0.919	0.922
SmallKitchenAppliances	0.725	0.796	**0.82**	0.753	0.814	0.817	0.813	0.803
SonyAIBORobotSurface1	0.938	**0.979**	0.918	0.887	0.916	0.902	0.903	0.969
SonyAIBORobotSurface2	0.909	0.959	0.918	**0.97**	0.962	0.961	0.964	0.953
StarLightCurves	0.974	0.976	**0.981**	0.977	0.98	0.98	**0.981**	**0.981**
Strawberry	0.977	0.973	0.981	0.982	0.983	**0.985**	0.983	0.981
SwedishLeaf	0.939	0.961	0.964	0.968	0.972	0.974	0.973	**0.975**

Table 2 (*continued*)

Datasets	EMAN	MACNN	ROCKET	ITime	LSTime	KDTime	KDCT	KDCR
Symbols	0.952	0.975	0.974	0.981	0.976	0.983	**0.984**	0.975
SyntheticControl	0.997	0.999	0.999	0.996	0.998	0.998	0.997	**1.0**
ToeSegmentation1	0.956	0.966	0.968	0.973	0.98	0.97	0.976	**0.988**
ToeSegmentation2	0.9	0.939	0.942	0.956	0.962	0.967	0.964	**0.969**
Trace	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**
TwoLeadECG	0.965	**0.999**	**0.999**	0.997	0.998	0.998	**0.999**	**0.999**
TwoPatterns	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**
UWaveGestureLibraryAll	0.966	0.956	**0.975**	0.894	0.898	0.897	0.897	0.899
UWaveGestureLibraryX	0.807	0.835	**0.85**	0.805	0.817	0.816	0.816	0.819
UWaveGestureLibraryY	0.728	**0.779**	0.773	0.728	0.749	0.741	0.737	0.744
UWaveGestureLibraryZ	0.75	0.784	**0.792**	0.761	0.77	0.767	0.768	0.77
Wafer	0.998	0.999	0.999	0.998	**1.0**	0.999	0.999	0.998
Wine	0.759	0.839	0.804	0.774	0.833	0.811	0.833	**0.926**
WordSynonyms	0.674	**0.758**	0.753	0.692	0.644	0.707	0.699	0.707
Worms	0.613	**0.849**	0.743	0.754	0.738	0.81	0.822	0.752
WormsTwoClass	0.757	0.859	0.79	0.808	0.891	0.884	0.9	**0.906**
Yoga	0.865	**0.916**	0.912	0.909	0.898	0.908	0.909	0.91
Total accuracy wins	13	26	22	6	8	9	18	30

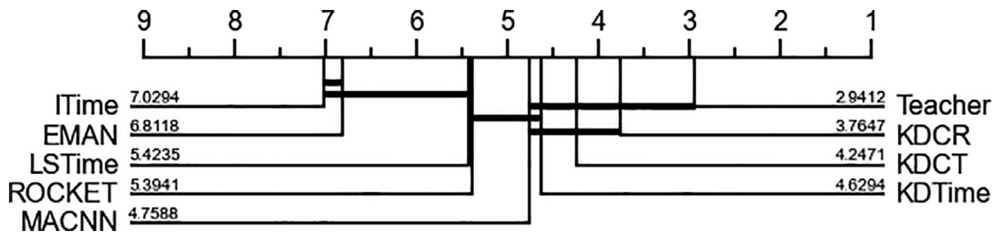


Fig. 12. Critical difference diagram illustrating the accuracy of different approaches.

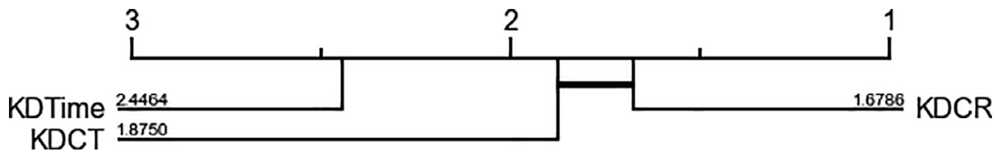


Fig. 13. Critical difference diagram illustrating the accuracy of three approaches on datasets with accuracy less than 0.8 for KDTime.

Fig. 14 demonstrates the training time of KDCTime compared with MACNN, EMAN, ROCKET, ITime, LSTime, and KDTime. Fig. 14(a) shows that KDCTime is much slower. In detail, the training time of KDCTime on the 62 datasets is below 1 order of magnitude slower than ROCKET, while that on the 23 datasets is more than 1 order of magnitude slower. This is because all approaches, except ROCKET, employ gradient descent as the optimizing algorithm, which requires the inference of many training samples for each update and a large number of updates in total, e.g., 64 samples for each update, many updating times in the 1 epoch, and 512 epochs in total. In contrast, ROCKET utilizes the ridge classifier and solves the ridge regression problem directly. However, the training time of KDCTime is still acceptable, which requires approximately 1 h to train 85 UCR datasets and 10 h for 10 times running in total. In addition, as shown in Fig. 14(b) and (c), the training time of ITime and LSTime is similar to that of KDCTime. The reason is that their model is the same, which is InceptionTime with 3 Inception modules. Nevertheless, KDCTime needs an extra teacher model to guide the training of its student model, which leads to the consequence that KDCTime requires extra training time for the teacher model. Note that training the teacher model is required only once. Once the teacher model is obtained, that model is available for multiple training iterations for the student model. Fig. 14(d) shows that the training time of KDCTime is not significantly different from KDTime; as their models are the same, they both require the teacher model, and gradient descent is selected as the optimization algorithm. Finally, MACNN and EMAN are relatively slow regarding training time, where EMAN is slightly faster. In conclusion, the difference in training time mainly appears in 3 categories of methods: ROCKET, ITime-based methods, and MACNN and EMAN.

Fig. 15 demonstrates the test time of KDCTime compared with MACNN, EMAN, ROCKET, ITime, LSTime, and KDTime. Fig. 15(a) shows that KDCTime is much faster than ROCKET. Specifically, the test time of KDCTime on 34 datasets is 1 order of magnitude faster than ROCKET, that on 41 datasets is 2 orders of magnitude faster, and that on 10 datasets is 3 orders of

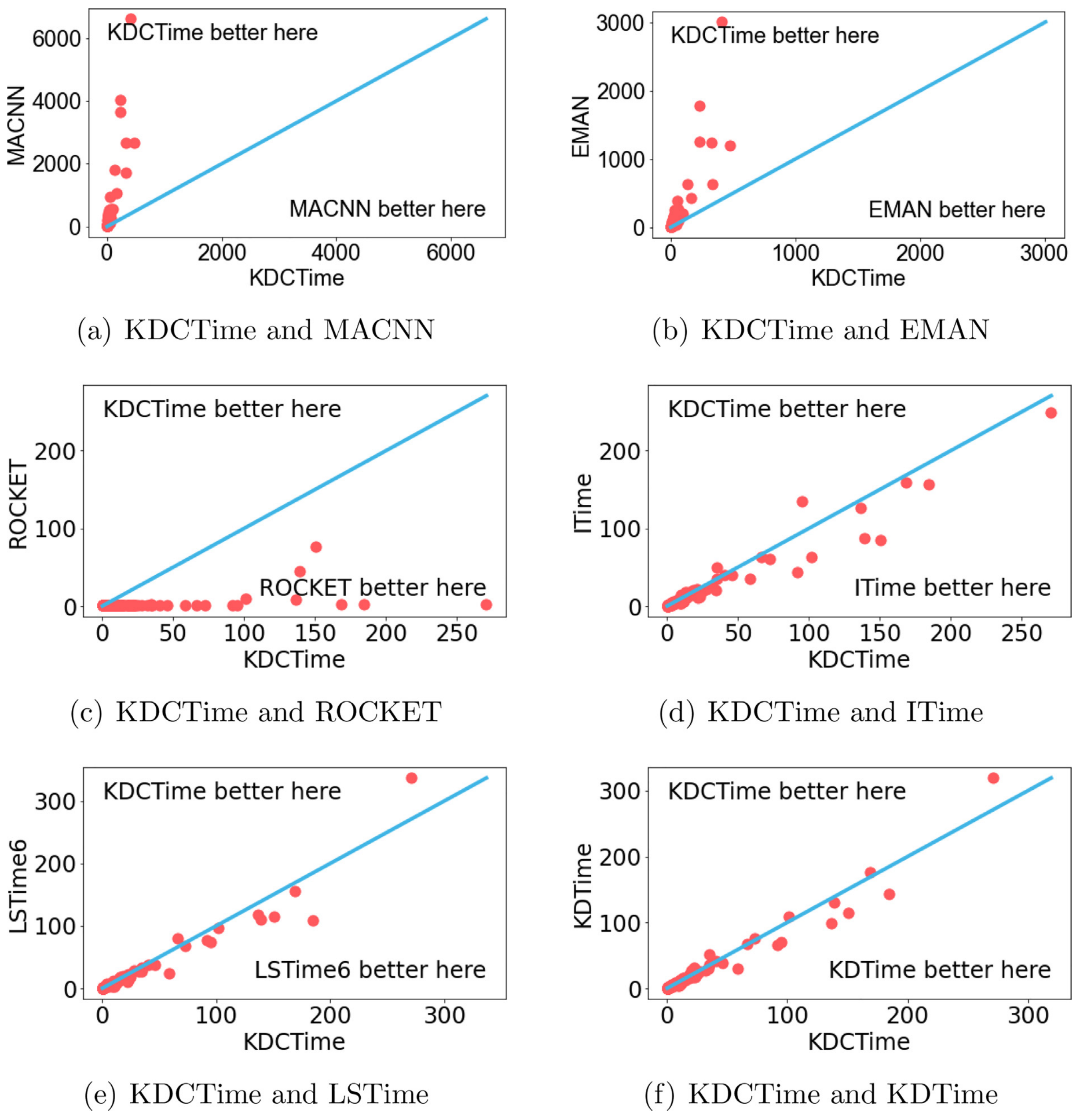


Fig. 14. The training time of KDCTime compared with MACNN, EMAN, ROCKET, ITime, LSTime, and KDTime.

magnitude faster. This is because in the test stage, 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 10,000 random convolutional kernels in ROCKET is much slower than that of the 3 inception modules in KDCTime. In addition, as shown in Fig. 15(b), (c), and (d), the test times of those approaches are not different from KDCTime since their inference models are all InceptionTime with 3 Inception modules. As a consequence, the difference in test time can be categorized into 2 groups, where the first group is MACNN, EMAN, and ROCKET, and the second group is ITime-based approaches.

4.6. Visualization of the features from the student and teacher models by t-SNE

To show the similarity of feature vectors between the student and teacher models, we adopt t-SNE to reduce the dimension of features to 2 and visualize them in Fig. 16. Since there are 85 datasets with different test samples and numbers of classes in the UCR, it is infeasible to visualize all. Thus, we randomly select 6 datasets to visualize, where the number of

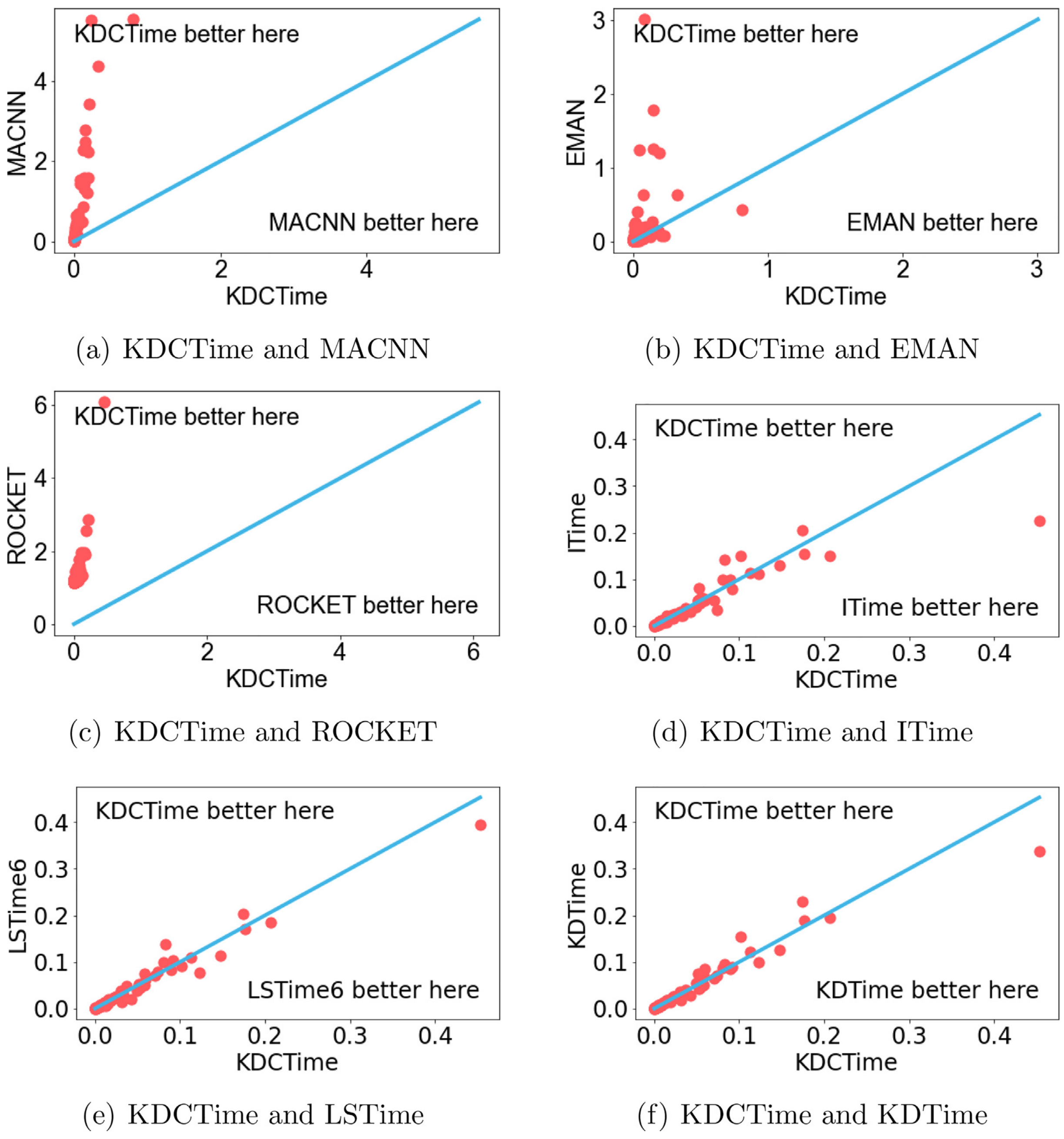


Fig. 15. The test time of KDCTime compared with MACNN, EMAN, ROCKET, ITime, LSTime, and KDTime.

classes ranges from 3 to 10. To be specific, they are ArrowHead with 175 samples and 3 classes, CinCECGTorso with 1,380 samples and 4 classes, Beef with 30 samples and 5 classes, Fish with 175 samples and 7 classes, Plane with 105 samples and 7 classes, and MedicalImages with 760 samples and 10 classes. Note that the perplexity of t-SNE is set to 30, and it is trained for 1,000 iterations.

In Fig. 16, points in different colors represent that they belong to distinct classes, where circles represent the feature vectors produced by the student model and stars represent the feature vectors produced by the teacher model. It is easy to observe that most points are grouped together. This means that feature vectors between the student and teacher models are relatively similar, and the knowledge from the teacher model is successfully distilled into the student model. However, there is still one exception, namely, the black points in Fig. 16(f). The black points belonging to the students are closer to the red and purple points, while the black points belonging to the teacher are far away. The reason for this is that the input time-series data corresponding to the black points are similar to the red and purple ones. Therefore, it is difficult to separate those

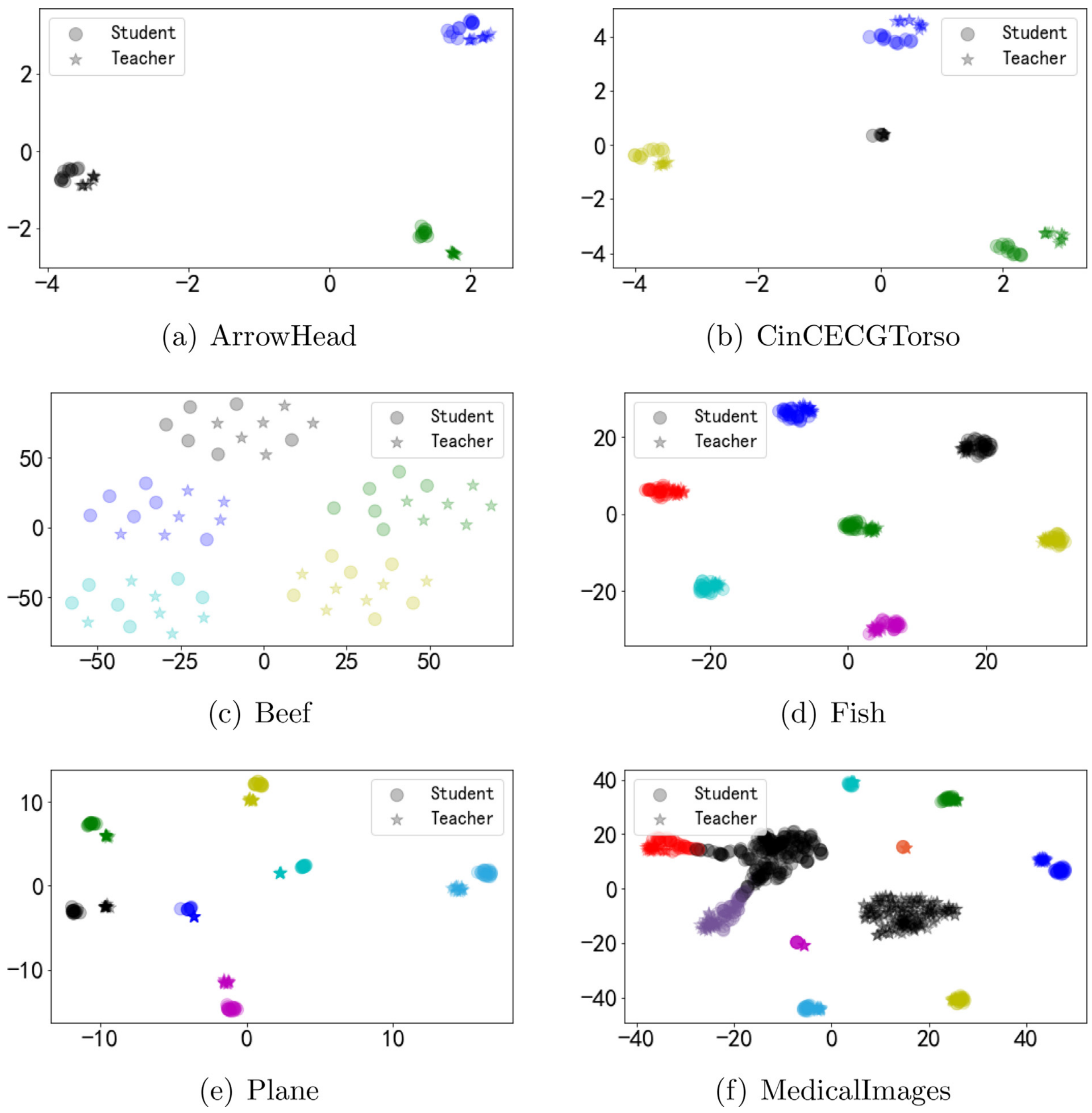
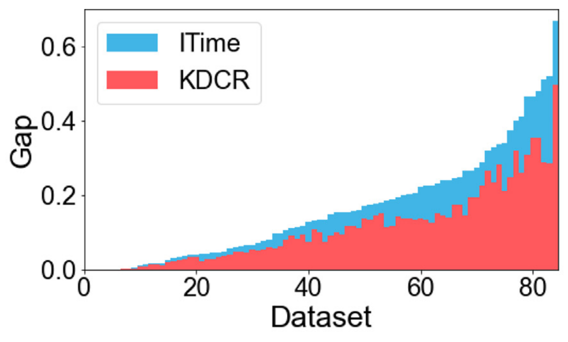


Fig. 16. Visualization of the features from the student and teacher models by t-SNE.

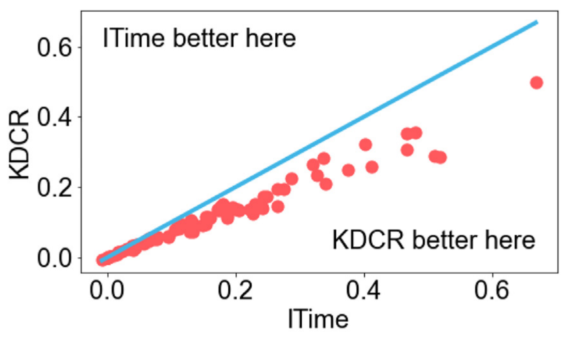
three classes for the student model. This indicates that knowledge distillation becomes harder when the number of classes increases, which will bring more groups of points for separation.

4.7. The capability of KDC for alleviating overfitting

In this section, we illustrate the training and test accuracy gap between ITime (without KDC) and KDCR (with KDC) to show the capability of KDC for alleviating overfitting. Note that only KDCR is compared since the KDCT accuracy is statistically lower than that of KDCR. As shown in Fig. 17(a), after sorting the datasets with respect to the training and test accuracy gap of ITime in ascending order, it is observed that KDCR reduces the training and test gap of ITime. In addition to the bar chart, we can also plot the comparison diagram to illustrate the reduction, as shown in Fig. 17(b), where the x-axis represents the gap of ITime, while the y-axis represents the gap of KDCR. In conclusion, KDC can alleviate the overfitting phenomenon on UCR datasets.



(a) The bar chart of training and test accuracy gap between ITime and KDCR



(b) The comparison diagram of training and test accuracy gap between ITime and KDCR

Fig. 17. The training and test accuracy gap between ITime and KDCR.

5. Conclusion

In this paper, we discovered that the DNN-based TSC approaches easily overfit UCR datasets, which is caused by the few-shot problem in the UCR archive. Thus, 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 to automatically generate soft labels. Finally, to rectify the incorrectly predicted soft labels from the teacher model, KD with calibration (KDC) was proposed, which has two optional strategies: KDC by translating (KDCT) and KDC by ordering (KDCR).

The experimental results show that the accuracies of KDCT and KDCR are promising, while KDCR obtains the highest accuracy. In addition, KDCR is much faster than MACNN and EMAN. For ROCKET, KDCR is 2 orders of magnitude faster than it is on test time. The training time of KDCR is slower than that of ROCKET, yet it is in an acceptable range and worthwhile to obtain a promising accuracy and fast inference time. Finally, KDCT and KDCR do not introduce any additional hyperparameters compared to ITime.

In the future, instead of only concentrating on the loss functions and labels, we will try various models to propose a brand new model that has a high generalization capability.

CRediT authorship contribution statement

Xueyuan Gong: Methodology, Software. **Yain-Whar Si:** Conceptualization, Validation. **Yongqi Tian:** Formal analysis. **Cong Lin:** Project administration. **Xinyuan Zhang:** Data curation. **Xiaoxiang Liu:** Supervision, Funding acquisition.

Declaration of Competing Interest

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

Acknowledgments

This work was supported in part by the Guangzhou Basic and Applied Basic Research Foundation under Grant 202201010476, the Fundamental Research Funds for the Central Universities under Grant 21621017, and the National Natural Science Foundation of China under Grant 62106085.

Appendix A. The procedure to calculate δ

Given a hard label $\mathbf{y}^h$ and a soft label $\mathbf{y}^t$, they both satisfy $\sum_{i=1}^{C} y_i = 1$ and $\forall i, y_i \geqslant 0$. 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 $\forall i, y_i^t \leqslant y_m^t$. Therefore, we have the following optimization objective:

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

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:

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

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

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

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

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

Finally, after solving this system of equations, we know that $\mathscr{L}(\mathbf{y}^t, \boldsymbol{\lambda}, \boldsymbol{\mu})$ 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 2.

References

[1] Anthony Bagnall, W.V. Jason Lines, E. Keogh, The uea & ucr time series classification repository, 2018. URL: http://www.timeseriesclassification.com/.
[2] A.J. Bagnall, J. Lines, A. Bostrom, J. Large, E.J. Keogh, The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances, Data Min. Knowl. Disc. 31 (2017) 606–660.
[3] A.J. Bagnall, J. Lines, J. Hills, A. Bostrom, Time-series classification with COTE: the collective of transformation-based ensembles, International Conference on Data Engineering (2016) 1548–1549.
[4] F.J. Baldan, J.M. Benítez, Multivariate times series classification through an interpretable representation, Inf. Sci. 569 (2021) 596–614.
[5] M.G. Baydogan, G.C. Runger, E. Tuv, A bag-of-features framework to classify time series, IEEE Trans. Pattern Anal. Mach. Intell. 35 (2013) 2796–2802.
[6] A. Benavoli, G. Corani, F. Mangili, Should we really use post-hoc tests based on mean-ranks?, J. Mach. Learn. Res. 17 (2016) 152–161.
[7] W. Chen, K. Shi, Multi-scale attention convolutional neural network for time series classification, Neural Networks 136 (2021) 126–140.
[8] J.H. Cho, B. Hariharan, On the efficacy of knowledge distillation, International Conference on Computer Vision (2019) 4793–4801.
[9] Z. Cui, W. Chen, Y. Chen, Multi-scale convolutional neural networks for time series classification, 2016. arXiv preprint arXiv:1603.06995.
[10] A. Dempster, F. Petitjean, G.I. Webb, ROCKET: exceptionally fast and accurate time series classification using random convolutional kernels, Data Min. Knowl. Disc. 34 (2020) 1454–1495.
[11] J. Demšar, Statistical comparisons of classifiers over multiple data sets, J. Mach. Learn. Res. 7 (2006) 1–30.
[12] J. Deng, W. Dong, R. Socher, L. Li, K. Li, L. Fei-Fei, Imagenet: A large-scale hierarchical image database, IEEE Conference on Computer Vision and Pattern Recognition (2009) 248–255.
[13] H.I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, P. Muller, Deep learning for time series classification: a review, Data Min. Knowl. Disc. 33 (2019) 917–963.
[14] H.I. Fawaz, B. Lucas, G. Forestier, C. Pelletier, D.F. Schmidt, J. Weber, G.I. Webb, L. Idoumghar, P. Muller, F. Petitjean, Inceptiontime: Finding alexnet for time series classification, Data Min. Knowl. Disc. 34 (2020) 1936–1962.

[15] M. Friedman, A comparison of alternative tests of significance for the problem of m rankings, Ann. Math. Stat. 11 (1940) 86–92.
[16] S. Garcia, F. Herrera, An extension on statistical comparisons of classifiers over multiple data sets for all pairwise comparisons, J. Mach. Learn. Res. 9 (2008).
[17] X. Gong, S. Fong, Y. Si, Fast multi-subsequence monitoring on streaming time-series based on forward-propagation, Inf. Sci. 450 (2018) 73–88.
[18] J. Gou, B. Yu, S.J. Maybank, D. Tao, Knowledge distillation: A survey, Int. J. Comput. Vision 129 (2021) 1789–1819.
[19] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, IEEE Conference on Computer Vision and Pattern Recognition (2016) 770–778.
[20] G. Hinton, O. Vinyals, J. Dean, Distilling the knowledge in a neural network, 2015. arXiv preprint arXiv:1503.02531.
[21] J. Hu, L. Shen, S. Albanie, G. Sun, E. Wu, Squeeze-and-excitation networks, IEEE Trans. Pattern Anal. Mach. Intell. 42 (2020) 2011–2023.
[22] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, International Conference on Machine Learning (2015) 448–456.
[23] S. Karimi-Bidhendi, F. Munshi, A. Munshi, Scalable classification of univariate and multivariate time series, International Conference on Big Data (2018) 1598–1605.
[24] R.J. Kate, Using dynamic time warping distances as features for improved time series classification, Data Min. Knowl. Disc. 30 (2016) 283–312.
[25] Y. Kowsar, M. Moshtaghi, E. Velloso, J.C. Bezdek, L. Kulik, C. Leckie, Shape-sphere: A metric space for analysing time series by their shape, Inf. Sci. 582 (2022) 198–214.
[26] A. Krizhevsky, Learning multiple layers of features from tiny images, 2009. URL: https://www.cs.utoronto.ca/~kriz/cifar.html.
[27] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86 (1998) 2278–2324.
[28] H. Li, Time works well: Dynamic time warping based on time weighting for time series data mining, Inf. Sci. 547 (2021) 592–608.
[29] H. Li, J. Liu, Z. Yang, R.W. Liu, K. Wu, Y. Wan, Adaptively constrained dynamic time warping for time series classification and clustering, Inf. Sci. 534 (2020) 97–116.
[30] J. Lines, S. Taylor, A.J. Bagnall, Time series classification with HIVE-COTE: the hierarchical vote collective of transformation-based ensembles, ACM Trans. Knowl. Discovery Data 12 (2018) 52:1–52:35.
[31] Q. Ma, Z. Zheng, W. Zhuang, E. Chen, J. Wei, J. Wang, Echo memory-augmented network for time series classification, Neural Networks 133 (2021) 177–192.
[32] S. Mirzadeh, M. Farajtabar, A. Li, N. Levine, A. Matsukawa, H. Ghasemzadeh, Improved knowledge distillation via teacher assistant, AAAI Conference on Artificial Intelligence (2020) 5191–5198.
[33] H. Oki, M. Abe, J. Miyao, T. Kurita, Triplet loss for knowledge distillation, International Joint Conference on, Neural Networks (2020) 1–7.
[34] S. Pei, T. Shen, X. Wang, C. Gu, Z. Ning, X. Ye, N. Xiong, 3dacn: 3d augmented convolutional network for time series data, Inf. Sci. 513 (2020) 17–29.
[35] T. Rakthanmanon, B.J.L. Campana, A. Mueen, G.E.A.P.A. Batista, M.B. Westover, Q. Zhu, J. Zakaria, 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.
[36] A. Romero, N. Ballas, S.E. Kahou, A. Chassang, C. Gatta, Y. Bengio, Fitnets: Hints for thin deep nets, in: International Conference on Learning Representations, 2015.
[37] Y. Sakurai, C. Faloutsos, M. Yamamuro, Stream monitoring under the time warping distance, International Conference on Data Engineering (2007) 1046–1055.
[38] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, J. Mach. Learn. Res. 15 (2014) 1929–1958.
[39] D. Svitov, S. Alyamkin, Margindistillation: Distillation for margin-based softmax, 2020. arXiv preprint arXiv:2003.02586.
[40] C. Szegedy, S. Ioffe, V. Vanhoucke, A.A. Alemi, Inception-v4, inception-resnet and the impact of residual connections on learning, AAAI Conference on Artificial Intelligence (2017) 4278–4284.
[41] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S.E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, IEEE Conference on Computer Vision and Pattern Recognition (2015) 1–9.
[42] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna, Rethinking the inception architecture for computer vision, IEEE Conference on Computer Vision and Pattern Recognition (2016) 2818–2826.
[43] Y. Wan, Y. Si, A formal approach to chart patterns classification in financial time series, Inf. Sci. 411 (2017) 151–175.
[44] Z. Wang, W. Yan, T. Oates, Time series classification from scratch with deep neural networks: A strong baseline, International Joint Conference on Neural Networks (2017) 1578–1585.
[45] T. Wen, S. Lai, X. Qian, Preparing lessons: Improve knowledge distillation with better supervision, Neurocomputing 454 (2021) 25–33.
[46] J. Yim, D. Joo, J. Bae, J. Kim, A gift from knowledge distillation: Fast optimization, network minimization and transfer learning, IEEE Conference on Computer Vision and Pattern Recognition (2017) 7130–7138.
[47] Z. Zheng, Z. Zhang, L. Wang, X. Luo, Denoising temporal convolutional recurrent autoencoders for time series classification, Inf. Sci. 588 (2022) 159–173.