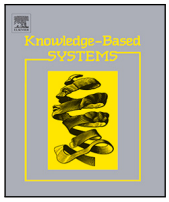
Knowledge-Based Systems 300 (2024) 112253

Contents lists available at ScienceDirect

Knowledge-Based Systems

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



POCKET: Pruning random convolution kernels for time series classification from a feature selection perspective

Shaowu Chen [a,b], Weize Sun [a,*], Lei Huang [a], Xiao Peng Li [a], Qingyuan Wang [b], Deepu John [b]

[a] State Key Laboratory of Radio Frequency Heterogeneous Integration (Shenzhen University), Shenzhen, 518060, China
[b] School of Electrical and Electronic Engineering, University College Dublin, Dublin 4, D04 V1W8, Ireland

ARTICLE INFO

Keywords:
Time series classification
Random convolution kernel
Pruning
Model compression
Feature selection

ABSTRACT

In recent years, two competitive time series classification models, namely, ROCKET and MINIROCKET, have garnered considerable attention due to their low training cost and high accuracy. However, they rely on a large number of random 1-D convolutional kernels to comprehensively capture features, which is incompatible with resource-constrained devices. Despite the development of heuristic algorithms designed to recognize and prune redundant kernels, the inherent time-consuming nature of evolutionary algorithms hinders efficient evaluation. To efficiently prune models, this paper eliminates feature groups contributing minimally to the classifier, thereby discarding the associated random kernels without direct evaluation. To this end, we incorporate both group-level ($l_{2,1}$-norm) and element-level (l_2-norm) regularizations to the classifier, formulating the pruning challenge as a group elastic net classification problem. An ADMM-based algorithm is initially introduced to solve the problem, but it is computationally intensive. Building on the ADMM-based algorithm, we then propose our core algorithm, POCKET, which significantly speeds up the process by dividing the task into two sequential stages. In Stage 1, POCKET utilizes dynamically varying penalties to efficiently achieve group sparsity within the classifier, removing features associated with zero weights and their corresponding kernels. In Stage 2, the remaining kernels and features are used to refit a l_2-regularized classifier for enhanced performance. Experimental results on diverse time series datasets show that POCKET prunes up to 60% of kernels without a significant reduction in accuracy and performs 11× faster than its counterparts. Our code is publicly available at https://github.com/ShaowuChen/POCKET.

1. Introduction

The time series classification (TSC) task has been widely explored in diverse application domains, including electrocardiogram diagnosis [1], incident detection [2], and fault locating [3]. In recent years, numerous advanced algorithms or models for TSC have emerged [4–7] to improve accuracy. However, a discernible trend is that the number of training parameters of algorithms is constantly increasing, incurring significant training time and computational resource overhead. This phenomenon becomes particularly conspicuous when dealing with large datasets.

The high complexity of various TSC methods has sparked an exploration of faster and more scalable alternatives [8–10]. The representative state-of-the-art works are ROCKET (for **R**and**O**m **C**onvolutional **KE**rnel **T**ransform) [11] and MINIROCKET [12], which leverage random 1-D convolutional kernels to extract features, namely, the proportion of positive values (PPV) and/or maximum values (MAX), which are then used to train linear classifiers. As the random kernels are training-free, ROCKET and MINIROCKET are more efficient in fitting classifiers compared to related TSC methods. Moreover, the utilization of PPV and MAX also reduces the dimension of the input data, reducing the complexity of the classifiers.

However, due to the randomness of convolution kernels, the extracted features may not be sufficiently discriminative for accurate classification. To comprehensively capture the temporal characteristics of time series data, ROCKET and MINIROCKET employ a large number of kernels, typically 10,000. Nevertheless, such an abundance of kernels introduces significant time and memory overhead during the inference process, making it challenging to deploy ROCKET and MINIROCKET on resource-constrained devices for real-time tasks. Additionally, the extracted features show a notable degree of redundancy [13], indicating that a substantial proportion of randomly generated kernels have a negligible impact on the TSC outcomes. Therefore, it becomes imperative

* Corresponding author.
E-mail addresses: shaowu-chen@foxmail.com (S. Chen), proton198601@hotmail.com (W. Sun), lhuang@szu.edu.cn (L. Huang), x.p.li@szu.edu.cn (X.P. Li), qingyuan.wang@ucdconnect.ie (Q. Wang), deepu.john@ucd.ie (D. John).

https://doi.org/10.1016/j.knosys.2024.112253
Received 5 February 2024; Received in revised form 23 June 2024; Accepted 13 July 2024
Available online 17 July 2024

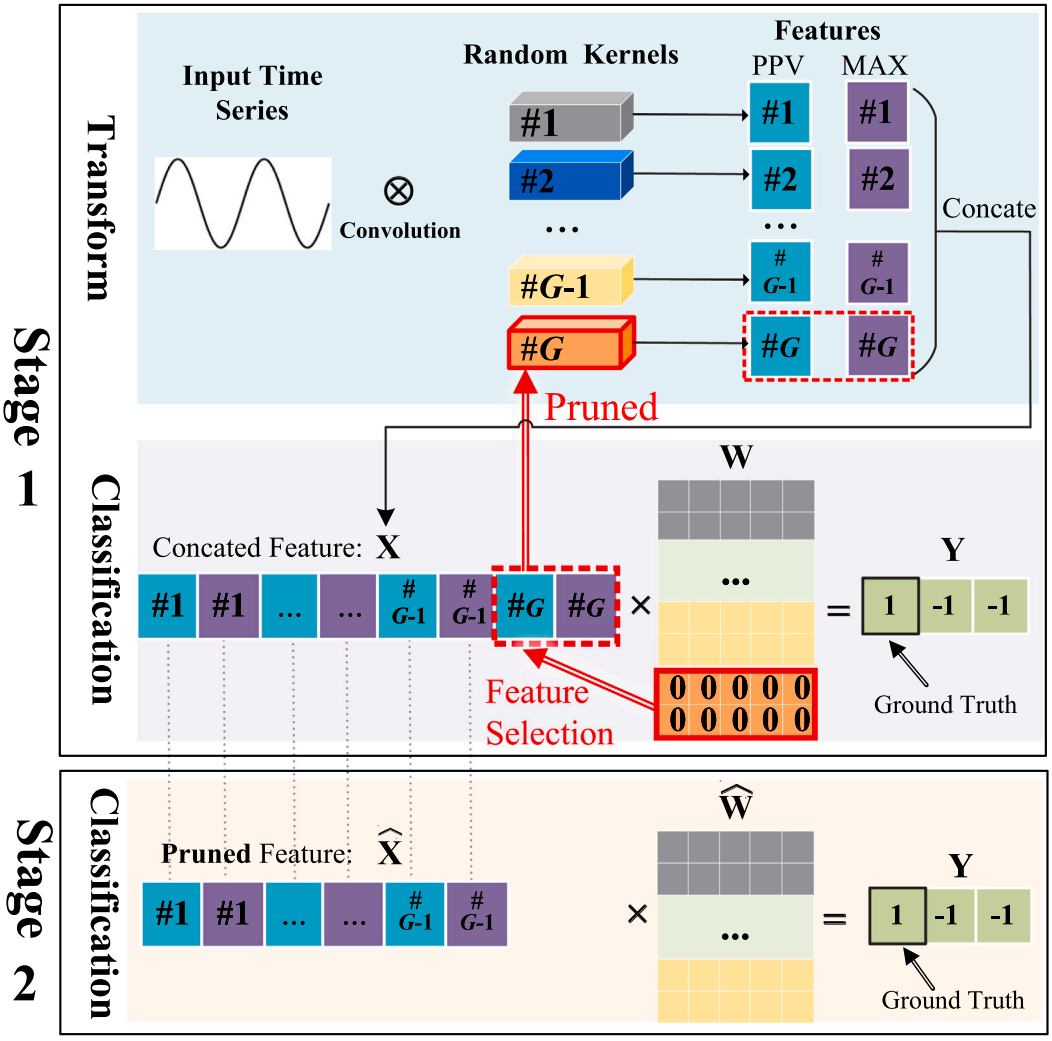


Fig. 1. Overall diagram of POCKET. **TOP:** Stage 1, remove feature groups that minimally contribute to the $\ell_{2,1}$-regularized classifier, thereby eliminating the corresponding kernels. **BOTTOM:** (optional) Stage 2, refit a ℓ_2-regularized classifier using the remaining (kernels and) features for enhanced performance.

to identify and prune redundant random kernels to enhance the efficiency of ROCKET and MINIROCKET while simultaneously preserving classification performance.

Selecting the optimal subset of kernels from the kernels of ROCKET and MINIROCKET can be considered a subset selection or a combinatorial optimization problem, whose goal is to maximize the accuracy of models while minimizing the number of kernels. Directly selecting kernels is NP-hard, due to the exponential number of possible combinations of kernels as the number of kernels increases, which makes it computationally infeasible to evaluate every possible subset to determine the optimal one. Additionally, as the random kernels are training-free, data-independent criteria such as ℓ_1 norm [14] and WHC [15] are less applicable. To tackle this challenge, Salehinejad et al. [13] introduced a differential evolutionary algorithm named S-ROCKET to effectively delete redundant kernels, albeit with a slight performance degradation. However, the evaluation period of S-ROCKET is time-consuming. Another potential drawback is that the pruning rate is determined by the evolutionary algorithm and cannot be predetermined based on available computational resources. This increases the risk of redundancy removal failure, where 100% of the kernels are retained, as observed in our experiment 4.3.

In contrast to existing approaches such as [13,16], which directly select or modify kernels, this paper proposes to efficiently prune the 'ROCKET family' through feature selection at the classification layer. The motivation behind this is that it is difficult to identify redundant kernels, as random kernels generated from the identical distribution do not exhibit significant differences, while the extracted features play notably distinguishable roles in the classification layer. As illustrated at the top of Fig. 1, if all the features extracted by a kernel minimally contribute to the subsequent classifier, the kernel can be considered redundant and safely removed without a substantial drop in accuracy. To achieve this, we formulate the pruning task as a $\ell_{2,1}$ constrained classification problem and incorporate ℓ_2 regularization to enhance model generalization, thereby transforming the challenge into a group elastic net problem. To solve this problem, we propose an ADMM-based algorithm that utilizes a dynamically varying soft threshold [17] during iterations, enabling a user-defined pruning rate. However, the iterative calculation of the inverse of a large matrix in each iteration makes the ADMM-based algorithm time-consuming.

To accelerate the solving process, we further introduce our core algorithm, **POCKET**, which stands for "**P**runing **R**OCKET**s**". Building on the initial ADMM-based algorithm, POCKET divides the $l_{2,1}$ and l_2 regularizations into two consecutive stages, solving them separately. As depicted at the top of Fig. 1, Stage 1 of POCKET prunes random kernels via feature selection by fitting a group-sparse classifier. It introduces relatively invariant penalties to avoid calculating the inverse in each iteration, significantly reducing the time complexity for achieving a solution. Stage 2, shown at the bottom of Fig. 1, is an optional phase that implements element-wise regularization to refit a high-accuracy

linear classifier using the remaining features from Stage 1. In our experiments, Stage 2 effectively mitigates the performance degradation of pruned MINIROCKET, which is less redundant and therefore sensitive to pruning.

The main contributions of this paper are summarized as follows.

1. We introduce a novel approach to pruning ROCKET models from a feature selection perspective, providing a more feasible alternative compared to the existing methods that directly evaluate random kernels.
2. The challenge of kernel pruning is addressed by formulating it as a group elastic net problem. We propose two algorithms to solve this problem, with the introduced POCKET showcasing an efficiency that is 11× faster than its counterparts. POCKET also allows for adjustable pruning rates in accordance with the computational budget.
3. Extensive experiments across various datasets are conducted to thoroughly evaluate the performance and efficacy of the proposed algorithms. The results demonstrate that when more than 60% of kernels on average are removed, the accuracy of ROCKET pruned by POCKET does not decrease significantly, but instead improves slightly.

The remainder of this paper is organized as follows. In Section 2, related works are summarized. Section 3 formulates the problem and introduces the proposed algorithms. In Section 4, experimental results are reported and discussed. Section 5 concludes the paper.

2. Related work

2.1. Time series classification

As TSC deals with inherently ordered data, algorithms for TSC are predominantly designed to capture distinctive order patterns of data, enabling effective classification. We categorize these approaches into six interrelated categories as outlined below, excluding the 'ROCKET family', which is elaborated on in the subsequent Section 2.2. For a more comprehensive introduction to TSC, readers can refer to relevant reviews [18,19].

Similarity based. This category usually leverages alignment metrics to measure similarity between time series data. To tackle the time-mismatch problem caused by various lengths of series in p-norm measure, elastic measures have emerged as promising alternatives to mitigate these limitations. For example, dynamic time warping (DTW) [20,21], complexity-invariant distance [22], and elastic ensemble [23] can evaluate the similarity of unaligned series and provide higher accuracy. Rather than calculating distance, pattern-based algorithms evaluate the similarity of trends or relative orders [24,25]. Motivated by OPR-Miner [25], which mines frequent patterns for classification, Wu et al. [26] proposed COPP-Miner to discover the most contrasting patterns, which are then used to train classifiers with superior performance.

Interval based. Rather than utilizing the entire time series for classification, interval-based methods split the time series into numerous intervals. The Time Series Forest (TSF) [27] constructs decision trees based on the statistical attributes of randomized intervals of the time series. In contrast, Random Interval Forest [28] selects random intervals and extracts spectral features, avoiding the use of statistical features. Using aggregated features extracted from randomly selected intervals, the randomized-supervised time series forest (r-STSF) [29] performs orders of magnitude faster than ensemble-based algorithm while being competitive in accuracy.

Shapelet based. Shapelet is a representative subsequence that encapsulates the fundamental characteristics of a dataset. Shapelet-based approaches assess the similarity between time series data and shapelets for TSC. Ye and Keogh [30] enumerated all conceivable candidates as shapelets and used an embedded decision tree structure. To accelerate the discovery of shapelets, the Fast Shapelets approach [31] uses a randomized projection technique to represent the symbolic aggregate approximation (SAX) of data. The Shapelet Transform [32] leverages optimal shapelets to evaluate distances and facilitate the training of classifiers. Contrastingly, Grabocka et al. [33] applied a gradient descent algorithm to a regularized logistic loss to learn shapelets, which are not present in the original series but are effective for TSC.

Dictionary based. Shapelet-based techniques may fail when class differentiation hinges on the relative occurrence of patterns rather than local subsequences. In such a scenario, dictionary-based methodologies [34,35] demonstrate efficacy. The Bag-of-SFA-Symbols (BOSS) algorithm [36] converts subsequences into 'words' using sliding windows and subsequently tally the frequency of recurring words. An evolution of BOSS, known as cBOSS [10], introduces a scalable variant. By amalgamating various dictionary methods, including cBOSS, the Temporal Dictionary Ensemble (TDE) [37] achieves state-of-the-art accuracy.

Ensemble based. Ensemble-based approaches use various classifiers to enhance performance. The HIVE-COTE algorithm [38] classifies data using a weighted average of multiple algorithms, including BOSS, Shapelet Transform, elastic distance measure, and frequency representation classifiers. Recently, various HIVE-COTE variants adapting TDE [37], Canonical Interval Forest (CIF) [39], and Diverse Representation CIF [40] have developed and shown significant improvement.

Deep Learning Based. The advancements in deep learning also contribute to the ongoing development of TSC [41–45]. InceptionTime [4] uses cascading inception modules [46] to extract important features from time series subsequences of various lengths. By combining handcrafted filters with InceptionTime, H-ITime [47] captures discriminate patterns across classes and achieves enhanced performance. To reduce the number of parameters, L-Time [48] employs custom convolutions and proposes a light-weight variant of InceptionTime.

2.2. Random kernel based models

Another type of TSC method is the 'ROCKET family' based on random kernels, including ROCKET [11], MINIROCKET [12], and their variants such as MultiROCKET [49].

The pioneering ROCKET model developed by Dempster et al. [11] utilizes a large number of 1-D random convolutional kernels, defaulting to 10,000, without the need for training. These kernels transform input time series into two sets of features, namely PPV and MAX, resulting in a total of 20,000 features, as depicted in Fig. 1. To fully capture features, the random kernels employed in ROCKET are parameterized with varying lengths, biases, dilations, and padding. Subsequently, a linear classifier is trained on the extracted features for classification, achieving state-of-the-art performance [50]. As the random kernels are train-free and only the classifier is fitted, the ROCKET model can efficiently train in a very short time.

MINIROCKET [12] is an improved variant of ROCKET, marked by two key distinctions. Firstly, MINIROCKET employs (almost) deterministic random kernels with binary values, such as 0 and 1, significantly accelerating convolution on samples. Secondly, MINIROCKET only utilizes the PPV feature, as the MAX feature is deemed less influential on classification accuracy. Consequently, the number of features is halved from 20,000 to 10,000. These enhancements also render MINIROCKET up to 75 times faster than ROCKET while maintaining competitive accuracy compared to other methods

Recently, a series of works building upon or drawing inspiration from ROCKET and MINIROCKET [34,49,51] have emerged. A common thread among these works is their continued reliance on a substantial number of random kernels to preserve the diversity of transformed features, albeit with potentially varying kernel configurations. Consequently, similar to ROCKET and MINIROCKET, these methodologies also contend with the challenge of excessive redundancy among random kernels, presenting obstacles to practical deployment.

Despite variations in the configurations of random kernels across different models, our proposed POCKET remains an effective approach for reducing redundancy. By disregarding the specifics of the kernels themselves and concentrating on pruning through feature selection within the classification layer, POCKET provides a generic solution to the common issue of kernel redundancy observed in various models relying on random kernels.

2.3. Model compression and S-ROCKET

Model compression has attracted significant attention in recent years, particularly for convolutional neural networks (CNNs). Four prominent strategies for compressing CNNs include low-rank factorization [52–55], quantization [56], neural architecture search [57], and pruning [58–60]. By considering the computation of PPV and MAX from convolutional feature maps as special pooling operations, ROCKET and MINIROCKET can be regarded as specialized shallow CNNs, each consisting of only one random hidden layer and one pooling layer. Therefore, it is reasonable to explore compressing ROCKET and MINIROCKET using CNN compression techniques, but the randomness of kernels makes the borrowing less straightforward.

Several works have specifically focused on pruning redundant random kernels within the 'Rocket family' [13,16,61]. S-ROCKET [13] is the representative pioneer that can prune around 60% of kernels with minor performance degradation following the three-step pipeline: pre-training, kernel selection, and post-training. However, the evolutionary algorithm used in the second step for kernel selection demands extensive iterations for evaluations and introduces a notable time overhead. Additionally, the pruning rate of S-ROCKET depends on the evaluation, introducing the potential risk of failure in pruning if a 0% pruning rate is encountered, as observed in our experiments in Section 4.3.2. In contrast, our proposed algorithms can achieve customized pruning rates according to the computational budget.

3. Methodology

In 3.1 and 3.2, we define notations and formulate the problem, respectively. Subsequently, our first ADMM-based algorithm is introduced in 3.3 to solve the problem. In 3.4, taking the ADMM-based algorithm as a prelude, we introduce our core two-stage algorithm, POCKET, to significantly accelerate the pruning process.

3.1. Notations

In this paper, italicized, bold lowercase, and bold uppercase letters are used to represent scalars, vectors, and matrices, respectively. The notation $\mathbf{I}_d$ represents the identity matrix of shape $d \times d$. The ith row, (i, j)th entry and trace of a matrix $\mathbf{A}$ are denoted as $\mathbf{A}_{i:}$, $\mathbf{A}_{i,j}$ and $\mathrm{tr}(\mathbf{A})$, respectively.

For simplicity, here we define the reshaped form of a matrix $\mathbf{A} \in \mathbb{R}^{I\times J}$ as $\mathbf{A}'$ with an operator reshape$(\cdot)$:

$$\mathbf{A}' := \mathrm{reshape}(\mathbf{A}) \tag{1}$$
$$= \begin{cases} \mathbf{A} \in \mathbb{R}^{I\times J}, & \text{in MINIROCKET,} \\ \begin{bmatrix} \mathbf{A}_{1:}, & \mathbf{A}_{2:} \\ \mathbf{A}_{3:}, & \mathbf{A}_{4:} \\ \cdots & \\ \mathbf{A}_{I-1:}, & \mathbf{A}_{I:} \end{bmatrix} \in \mathbb{R}^{\frac{I}{2}\times 2J}, & \text{in ROCKET.} \end{cases}$$

3.2. Problem formulation

Different from S-ROCKET [13], which focuses on the convolutional layer with random weights, we aim to prune redundant kernels effectively by eliminating related features in the classifier, as illustrated in Fig. 1. To this end, we impose group-level or row-wise sparsity in the classifiers for ROCKET and MINIROCKET, respectively:

$$\begin{aligned} \min_{\mathbf{W}} \quad & \|\mathbf{W}'\|_{2,0} \\ \text{s.t.} \quad & \|\mathbf{W}\|_F < c_1, \quad \mathbf{XW} = \mathbf{Y}. \end{aligned} \tag{2}$$

Here, $\mathbf{W} \in \mathbb{R}^{H\times C}$ is the weight matrix for C-class classification, $\mathbf{W}' = \mathrm{reshape}(\mathbf{W})$ as defined in Eq. (1), and c_1 is a user-defined constant that limits the magnitude of $\mathbf{W}$ and enhances the generalization of the classifier. $\mathbf{X} \in \mathbb{R}^{N\times H}$ represents the feature matrix constituted by H features extracted by G random convolution kernels from N samples, where $G := H$ for MINIROCKET and $G := \frac{H}{2}$ for ROCKET, respectively. (Note that each kernel of ROCKET extracts PPV and MAX, leading to two consecutive rows within $\mathbf{X}$ originating from a single kernel; whereas MINIROCKET solely extracts PPV.) Here $\mathbf{Y} \in \mathbb{R}^{N\times C}$ corresponds to the class indicator. The ground-truth $\mathbf{Y}$ is binary-coded (± 1), so that the only positive element in each row indicates the class of the corresponding time series data. This manner effectively achieves C-class classification using C two-category classifiers based on regression. During the prediction phase, $\mathbf{XW}$ is usually not strictly equal to the ground-truth $\mathbf{Y}$, and the categories of samples are indicated by the indices of the largest elements within each row of $\mathbf{XW}$.

Solving the $\ell_{2,0}$ problem (2) is NP-hard, therefore, we reformulate our objective as a group elastic net problem:

$$\begin{aligned} \min_{\mathbf{W}} \quad & \|\mathbf{W}'\|_{2,1} \\ \text{s.t.} \quad & \|\mathbf{W}\|_F < c_1, \quad \|\mathbf{XW} - \mathbf{Y}\|_F < c_2, \end{aligned} \tag{3}$$

which is convex and easier to solve. Here, $\|\mathbf{W}'\|_{2,1}$ is the convex approximation of $\|\mathbf{W}'\|_{2,0}$. We substitute the strict constraint $\mathbf{XW} = \mathbf{Y}$ with the relaxed $\|\mathbf{XW} - \mathbf{Y}\|_F < c_2$, where c_2 is a hyperparameter, to improve the noise tolerance of the classifier.

3.3. Prelude: ADMM-based algorithm

In this subsection, we propose an ADMM-based algorithm to solve the problem (3). Based on the algorithm, we will introduce the core two-stage pruning algorithm, the accelerated POCKET, in the next subsection.

To utilize ADMM [58,62], we rewrite problem (3) as

$$\begin{aligned} \min_{\mathbf{W},\boldsymbol{\Theta}} \quad & \|\boldsymbol{\Theta}'\|_{2,1} \\ \text{s.t.} \quad & \|\mathbf{W}\|_F < c_1, \\ & \|\mathbf{XW} - \mathbf{Y}\|_F < c_2, \\ & \boldsymbol{\Theta} = \mathbf{W}, \end{aligned} \tag{4}$$

where $\boldsymbol{\Theta}' = \mathrm{reshape}(\boldsymbol{\Theta})$. Here and in the subsequent sections, we assume that rows of $\mathbf{X}$ are zero-centered and scaled to have unit ℓ_2 norms, and the $\mathbf{Y}$ is column-wise zero-centered. By applying the penalty method to the inequality constraints and attaching the Lagrangian multiplier to the equality constraint, we formulate the augmented Lagrangian function for the problem (4) as

$$\begin{aligned} \mathrm{L}(\mathbf{W}, \boldsymbol{\Theta}, \boldsymbol{\Lambda}) = & \|\boldsymbol{\Theta}'\|_{2,1} + \frac{\rho_1}{2}\|\mathbf{W}\|_F^2 + \frac{\rho_2}{2}\|\mathbf{XW} - \mathbf{Y}\|_F^2 \\ & + \frac{\rho_3}{2}\|\boldsymbol{\Theta} - \mathbf{W}\|_F^2 + \mathrm{tr}\left(\boldsymbol{\Lambda}^T \circ (\boldsymbol{\Theta} - \mathbf{W})\right), \end{aligned} \tag{5}$$

where $\boldsymbol{\Lambda} \in \mathbb{R}^{H\times C}$ denotes the Lagrange multiplier, and '$\circ$' signifies element-wise multiplication. Here ρ_1, ρ_2, and ρ_3 are positive penalty scalars, which are user-defined hyperparameters. Generally, a smaller value of c_1 leads to a larger ρ_1, and the same relationship holds for c_2 and ρ_2.

Leveraging ADMM [62], problem (4) can be solved by iteratively updating the following variables:

$$\mathbf{W}^{(t)} = \arg\min_{\mathbf{W}} \quad \mathrm{L}\left(\mathbf{W}, \boldsymbol{\Theta}^{(t-1)}, \boldsymbol{\Lambda}^{(t-1)}\right), \tag{6}$$

$$\mathbf{\Theta}^{(t)} = \arg\min_{\mathbf{\Theta}} \quad \mathrm{L}\left(\mathbf{W}^{(t)}, \mathbf{\Theta}, \mathbf{\Lambda}^{(t-1)}\right), \tag{7}$$

$$\mathbf{\Lambda}^{(t)} = \mathbf{\Lambda}^{(t-1)} + s * \left(\mathbf{\Theta}^{(t)} - \mathbf{W}^{(t)}\right), \tag{8}$$

where t signifies the step of iteration, and $s > 0$ is the step size for updating the multiplier.

3.3.1. Updating W

By defining

$$\mathbf{U} = \frac{\mathbf{\Lambda}}{\rho_3}, \tag{9}$$

the problem (6) can be formulated as

$$\begin{aligned}\mathbf{W}^{(t)} &= \arg\min_{\mathbf{W}} \frac{\rho_1}{2}\|\mathbf{W}\|_F^2 + \frac{\rho_2}{2}\|\mathbf{X}\mathbf{W} - \mathbf{Y}\|_F^2 \\ &\quad + \frac{\rho_3}{2}\|\mathbf{\Theta}^{(t-1)} - \mathbf{W}\|_F^2 + \mathrm{tr}\left(\{\mathbf{\Lambda}^{(t-1)}\}^T \circ (\mathbf{\Theta}^{(t-1)} - \mathbf{W})\right) \\ &= \arg\min_{\mathbf{W}} \frac{\rho_1}{2}\|\mathbf{W}\|_F^2 + \frac{\rho_2}{2}\|\mathbf{X}\mathbf{W} - \mathbf{Y}\|_F^2 \\ &\quad + \frac{\rho_3}{2}\|\mathbf{\Theta}^{(t-1)} - \mathbf{W} + \mathbf{U}^{(t-1)}\|_F^2.\end{aligned} \tag{10}$$

By setting the gradient with respect to $\mathbf{W}$ to zero,

$$\rho_1\mathbf{W} + \rho_2\mathbf{X}^T\mathbf{X}\mathbf{W} - \rho_2\mathbf{X}^T\mathbf{Y} + \rho_3\mathbf{W} - \rho_3(\mathbf{\Theta}^{(t-1)} + \mathbf{U}^{(t-1)}) = 0, \tag{11}$$

we arrive at

$$\mathbf{W}^{(t)} = \left[(\rho_1 + \rho_3)\mathbf{I}_F + \rho_2\mathbf{X}^T\mathbf{X}\right]^{-1}\left[\rho_3\left(\mathbf{\Theta}^{(t-1)} + \mathbf{U}^{(t-1)}\right) + \rho_2\mathbf{X}^T\mathbf{Y}\right]. \tag{12}$$

In the next subsection, we will introduce a dynamic $\rho_3^{(t)}$ to achieve a predetermined pruning rate. Consequently, Eq. (12) will be modified to

$$\mathbf{W}^{(t)} = [(\rho_1 + \rho_3^{(t)})\mathbf{I}_F + \rho_2\mathbf{X}^T\mathbf{X}]^{-1}\left[\rho_3^{(t)}\left(\mathbf{\Theta}^{(t-1)} + \mathbf{U}^{(t-1)}\right) + \rho_2\mathbf{X}^T\mathbf{Y}\right]. \tag{13}$$

3.3.2. Updating Θ

The updated $\mathbf{\Theta}$ is

$$\begin{aligned}\mathbf{\Theta}^{(t)} &= \arg\min_{\mathbf{\Theta}} \|\mathbf{\Theta}'\|_{2,1} + \frac{\rho_3}{2}\|\mathbf{\Theta} - \mathbf{W}^{(t)}\|_F^2 \\ &\quad + \mathrm{tr}\left(\{\mathbf{\Lambda}^{(t-1)}\}^T \circ \left(\mathbf{\Theta} - \mathbf{W}^{(t)}\right)\right) \\ &= \arg\min_{\mathbf{\Theta}} \|\mathbf{\Theta}'\|_{2,1} + \frac{\rho_3}{2}\|\mathbf{\Theta} - \mathbf{W}^{(t)} + \mathbf{U}^{(t-1)}\|_F^2,\end{aligned} \tag{14}$$

whose grouped components can be obtained using the soft-threshold operator $\mathrm{S}(\cdot)$ [17] with the threshold $\frac{1}{\rho_3}$:

$$\begin{aligned}\mathbf{\Theta}_{g:}'^{(t)} &= \mathrm{S}_{\frac{1}{\rho_3}}\left(\mathbf{W}_g'^{(t)} - \mathbf{U}_g'^{(t-1)}\right) \\ &= \left(\mathbf{W}_g'^{(t)} - \mathbf{U}_g'^{(t-1)}\right) * \max\left(1 - \frac{\frac{1}{\rho_3}}{\|\mathbf{W}_g'^{(t)} - \mathbf{U}_g'^{(t-1)}\|_2}, 0\right),\end{aligned} \tag{15}$$

following the convention that $\frac{0}{0} = 1$, where $g = 1, 2, \ldots, G$.

In Eq. (15), the parameter ρ_3 controls the penalty imposed on the magnitude of groups in Θ. However, determining a suitable value for ρ_3 to achieve the desired pruning rate is a challenging task that often requires multiple attempts. To address this issue, motivated by [17], we adopt a dynamic threshold, $\rho_3^{(t)}$, to allow for predetermining the number of remaining kernels m (where $m < G$). Denoting $\widetilde{\mathbf{W}}'^{(t)} = \mathbf{W}'^{(t)} - \mathbf{U}'^{(t-1)}$, we define the dynamic threshold as

$$\frac{1}{\rho_3^{(t)}} = \|\mathbf{W}'^{(t)}_{f_{m+1}:} - \mathbf{U}'^{(t-1)}_{f_{m+1}:}\|_2 = \|\widetilde{\mathbf{W}}'^{(t)}_{f_{m+1}:}\|_2, \tag{16}$$

which satisfies

$$\|\widetilde{\mathbf{W}}'^{(t)}_{f_1:}\|_2 \geq \cdots \geq \|\widetilde{\mathbf{W}}'^{(t)}_{f_m:}\|_2 \geq \|\widetilde{\mathbf{W}}'^{(t)}_{f_{m+1}:}\|_2 \geq \cdots \geq \|\widetilde{\mathbf{W}}'^{(t)}_{f_G:}\|_2, \tag{17}$$

where $(f_1, \cdots, f_m, f_{m+1}, \ldots, f_G)$ is a permutation of $(1, 2, \ldots, G)$. Using the dynamic threshold, the update for the grouped components of $\mathbf{\Theta}$ is

$$\mathbf{\Theta}_g'^{(t)} = \widetilde{\mathbf{W}}'^{(t)}_g * \max\left(1 - \frac{\|\widetilde{\mathbf{W}}'^{(t)}_{f_{m+1}:}\|_2}{\|\widetilde{\mathbf{W}}'^{(t)}_g\|_2}, 0\right), \tag{18}$$

for $g = 1, 2, \ldots, G$.

3.3.3. Updating scale-form dual variable

Dividing the both sides of Eq. (8) by the step size s, we have

$$\frac{\mathbf{\Lambda}^{(t)}}{s} = \frac{\mathbf{\Lambda}^{(t-1)}}{s} + \left(\mathbf{\Theta}^{(t)} - \mathbf{W}^{(t)}\right). \tag{19}$$

Setting $s = \rho_3$, we arrive at

$$\mathbf{U}^{(t)} = \mathbf{U}^{(t-1)} + \mathbf{\Theta}^{(t)} - \mathbf{W}^{(t)}, \tag{20}$$

which is the updating equation for the scale-form dual variable matrix $\mathbf{U}$ defined in (9). Since the iteration steps (13) and (18) involve $\mathbf{U}$ rather than $\mathbf{\Lambda}$, we can directly update the scaled dual variable $\mathbf{U}$ during the iterations.

3.3.4. Summary and time complexity

By iteratively updating (13), (16), (18), and (20), we can successfully solve the random kernel pruning problem from a feature selection perspective. The dynamic threshold described in (16) allows for arbitrary desired pruning rates according to the computational budget.

The disadvantage of the ADMM-based algorithm is its high computational complexity, $\mathcal{O}(TH^3)$ (assuming $H > N$), where T represents the number of iterations. As $\rho_3^{(t)}$ is dynamically varying, updating (13) requires the calculation of a large matrix inverse in each iteration. With H being 20K and 10K for ROCKET and MINIROCKET, respectively, it is time-consuming to prune redundant convolutional kernels.

3.4. Accelerated algorithm: POCKET

To accelerate the ADMM-based algorithm, we further propose our core algorithm, POCKET. In contrast to the ADMM-based algorithm, POCKET employs a two-stage strategy that separates group-level and element-wise regularization. As illustrated in Fig. 1, the first stage implements group-wise regularization for pruning, and the optional second stage regularizes element-wise magnitudes to refit a high-accuracy classifier.

3.4.1. Stage 1 (Group-wise Feature Selection)

In the first stage, we only implement group-wise feature selection by solving

$$\begin{aligned}&\min_{\mathbf{W},\mathbf{\Theta}} \quad \|\mathbf{\Theta}'\|_{2,1} \\ &\text{s.t.} \quad \|\mathbf{X}\mathbf{W} - \mathbf{Y}\|_F < c_2^{(t)}, \quad \mathbf{\Theta} = \mathbf{W}.\end{aligned} \tag{21}$$

Note that, different from problem (4), here we temporarily omit the constraint $\|\mathbf{W}\|_F < c_1$. Furthermore, we allow the penalty strength on $\|\mathbf{X}\mathbf{W} - \mathbf{Y}\|_F$ to dynamically vary in each iteration step, as indicated by the superscript (t) on c_2. The problem can be solved using the ADMM-based algorithm proposed in the last subsection by simply replacing the update equation of $\mathbf{W}$ in (13) with

$$\mathbf{W}^{(t)} = \left[\rho_3^{(t)}\mathbf{I}_F + \rho_2^{(t)}\mathbf{X}^T\mathbf{X}\right]^{-1}\left[\rho_3^{(t)}\left(\mathbf{\Theta}^{(t)} + \mathbf{U}^{(t-1)}\right) + \rho_2^{(t-1)}\mathbf{X}^T\mathbf{Y}\right]. \tag{22}$$

Here, the adaptive $\rho_3^{(t)}$ is the reciprocal of the soft threshold (16), while $\rho_2^{(t)}$ is associated with the dynamically changing $c_2^{(t)}$.

However, the high computational complexity of (22) remains, since the computation of $[\rho_3^{(t)}\mathbf{I}_f + \rho_2^{(t)}\mathbf{X}^T\mathbf{X}]^{-1}$ is required in each iteration. To address the issue, we further introduce a requirement to maintain the relative invariance between $\rho_2^{(t)}$ and $\rho_3^{(t)}$:

$$\frac{\rho_3^{(t)}}{\rho_2^{(t)}} = k, \tag{23}$$

where $k > 0$ is a user-defined constant. In this scenario, Eq. (22) can be rewritten as

$$\mathbf{W}^{(t)} = (k\mathbf{I}_F + \mathbf{X}^T\mathbf{X})^{-1}\left[k\left(\mathbf{\Theta}^{(t-1)} + \mathbf{U}^{(t-1)}\right) + \mathbf{X}^T\mathbf{Y}\right]. \tag{24}$$

Here, the inverse term $(k\mathbf{I}_f+\mathbf{X}^T\mathbf{X})^{-1}$ remains constant during iteration, thus it can be precomputed once and stored for reuse, leading to a significant reduction in time complexity. Given the relatively large feature dimensions of ROCKET and MINIROCKET (20K and 10K, respectively), the Sherman–Morrison–Woodbury formula [63,64] can be employed to further expedite the calculation for small-sample datasets ($N \ll H$):

$$(k\mathbf{I}_F + \mathbf{X}^T\mathbf{X})^{-1} = \frac{1}{k}\left[\mathbf{I}_F - \mathbf{X}^T(k\mathbf{I}_N + \mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}\right], \tag{25}$$

which reduces the complexity from $\mathcal{O}(H^3)$ to $\mathcal{O}(N^3)$.

By repeating (24), (16), (18) and (20), problem (21) can be solved to result in a group-sparse classifier that prunes redundant kernels via feature selection. The weight groups in $\mathbf{W}$ with zeros values and their corresponding convolution kernels can be safely deleted to reduce the overall size of the model. However, the weights in $\mathbf{W}$ that should be pruned often remain very small magnitudes. As $\mathbf{\Theta}$ in (15) possesses exactly zero-value groups and $\mathbf{W}$ should converge to $\mathbf{\Theta}$, we eliminate the groups in $\mathbf{W}$ that correspond to the zero-valued groups of $\mathbf{\Theta}$. Removing groups with very small values in $\mathbf{W}$ has a negligible impact on classification since the classifier fitted by $\|\mathbf{X}\mathbf{W}-\mathbf{Y}\|_F < c_2$ is noise-tolerant.

3.4.2. Stage 2 (Optional, element-wise Regularization)

In Stage 2, we implement the element-wise regularization and refit a ridge classifier using the features selected by Stage 1:

$$\begin{aligned} \min_{\widehat{\mathbf{W}}} \quad & \|\widehat{\mathbf{X}}\widehat{\mathbf{W}} - \mathbf{Y}\|_F \\ \text{s.t.} \quad & \|\widehat{\mathbf{W}}\|_F < c_1, \end{aligned} \tag{26}$$

where $\widehat{\mathbf{X}} \in \mathbb{R}^{N\times\hat{H}}$ signifies the feature matrix after pruning by Stage 1, $\widehat{\mathbf{W}} \in \mathbb{R}^{\hat{H}\times C}$ represents the weight matrix, and $\hat{H} \in \{1,2,\dots,H-1\}$ denotes the number of remaining features. The solution of (26) can be written as

$$\widehat{\mathbf{W}} = \left(\rho_1\mathbf{I} + \widehat{\mathbf{X}}^T\widehat{\mathbf{X}}\right)^{-1}\widehat{\mathbf{X}}^T\mathbf{Y}, \tag{27}$$

where ρ_1 is a user-determined hyperparameter.

Since Stage 1 has already trained a classifier during feature selection, Stage 2 is optional and is primarily useful for addressing potential overfitting issues. In experiments, we observe that MINIROCKET is more sensitive to pruning as it is less redundant, and thus Stage 2 with element-wise regularization is more useful for MINIROCKET to significantly improve classification performance.

3.4.3. Summary

We summarize our core algorithm, POCKET, in Algorithm 1. Different from the ADMM-based algorithm proposed in Section 3.3, POCKET divides the group-wise and element-wise regularizations into two sequential stages. The first stage performs kernel pruning via feature selection by imposing group sparsity in the classifier. The optional second stage uses ridge regularization to refit a classifier to alleviate potential overfitting.

The advantage of POCKET compared to the ADMM-based algorithm is its low time complexity. In Stage 1, by maintaining relative invariance between $\rho_2^{(t)}$ and $\rho_3^{(t)}$, POCKET computes the inverse of the large matrix only once. Except for the two one-time inverse computations in (24) and (27), there are only matrix multiplies in iterations, which greatly reduces the time complexity.

Algorithm 1: POCKET

Input:
Time series data matrix $\mathbf{D}$ with N samples and label matrix $\mathbf{Y}$.
Number of remained kernels m, number of iteration T.
Hyper-parameters k and ρ_1. (In practice, they can be selected by cross-validation.)
Output:
The lightweight ROCKET/MINIROCKET model with m kernels and a classifier $\widehat{\mathbf{W}}$.

// **Initialization:**
Generate G random convolution kernels and extract the feature matrix $\mathbf{X} \in \mathbb{R}^{N\times H}$ from $\mathbf{D}$.
Normalize $\mathbf{X}$ to make each group of features have a zero center and unit norm.
Centralize each column of $\mathbf{Y}$ separately.

// **Stage 1**: Pruning via Group Feature Selection
Precompute $(k\mathbf{I}_f + \mathbf{X}^T\mathbf{X})^{-1}$. (Use (25) if $N \ll H$.)
$\mathbf{\Theta} \leftarrow \mathbf{0}$, $\mathbf{U} \leftarrow \mathbf{0}$.
for $t = 1, 2, \cdots, T$ **do**
  Compute $\mathbf{W}$ by (24).
  Update ρ_3 by (16).
  Compute $\mathbf{\Theta}$ by (18).
  Update $\mathbf{U}$ by (20).
end
Remain m groups of weight in $\mathbf{W}$ indicated by nonzero components of $\mathbf{\Theta}$ and delete the others to get $\widehat{\mathbf{W}}$; delete the associated redundant random convolution kernels and features in $\mathbf{X}$ to get $\widehat{\mathbf{X}}$.

// **Stage 2** (Optional): Element-wise Regularization
Recompute $\widehat{\mathbf{W}}$ by (27).

return m remained random kernels, $\widehat{\mathbf{W}}$.

4. Experiments

4.1. Data, models and general settings

4.1.1. Datasets

Following [11,13], we evaluate our algorithms on the UCR archive [66], which contains 128 time series datasets, including 85 'bake off' datasets and 43 'extra' ones. These datasets cover a wide range of application domains and serve as a comprehensive benchmark for evaluating time series classification algorithms. For missing values of data samples, we simply fill them with zeros.

4.1.2. Target models

In line with S-ROCKET [13], we prune MINIROCKET and ROCKET trained with only the PPV feature. However, to provide a more comprehensive evaluation, we also prune the original full ROCKET model trained with both PPV and MAX features. To differentiate between the full ROCKET model with both MAX and PPV features, and the one with only PPV, we will refer to the former as '**ROCKET**' or '**ROCKET-PPV-MAX**', and the latter as '**ROCKET-PPV**'.

4.1.3. General settings

Unless explicitly stated, all experiments are conducted using Python 3.6.12 and scikit-learn 0.24.2 [67] on a Ubuntu 18.04.6 computer with two Intel Xeon E5-2690 CPUs and 56 threads in total. For a fair comparison, we use the official code and settings to reproduce ROCKET, MINIROCKET, and S-ROCKET.

For our ADMM-based algorithm and POCKET, the number of iterations for both cross-validation and fitting is $T = 50$. Hyper-parameters ρ_1 and ρ_2 for the ADMM-based algorithm, and k for Stage 1 of POCKET, are determined through 5-fold cross-validations. Following ROCKET [13] and MINIROCKET [12], ρ_1 in Stage 2 of POCKET is decided by the efficient Leave-One-Out cross-validation on the log scale range of $[-3, 3]$. Each evaluation is repeated 10 times to report the average results. More implementation details can be found in our publicly available code.[1]

[1] https://github.com/ShaowuChen/POCKET.

Table 1
Testing accuracy comparison on pruning 50% kernels of ROCKET-PPV-MAX. The '**Ave. w/o Beef**' row presents the average testing accuracy across datasets except for the 'Beef'.

Dataset	Original ROCKET (%)	ADMM-based pruning (%)	POCKET (%)	
			Stage 1	Stage 2
Adiac	78.44 ± 0.61	81.18 ± 0.86	80.84 ± 0.98	80.23 ± 0.67
ArrowHead	81.20 ± 0.90	81.89 ± 1.20	81.54 ± 1.58	80.86 ± 0.82
Beef	83.33 ± 1.49	51.00 ± 2.60	83.67 ± 1.00	84.33 ± 1.53
BeetleFly	90.00 ± 0.00	90.00 ± 0.00	90.00 ± 0.00	90.00 ± 0.00
BirdChicken	90.00 ± 0.00	90.00 ± 0.00	90.00 ± 0.00	90.00 ± 0.00
Car	86.00 ± 3.35	91.5 ± 1.38	89.83 ± 4.74	90.50 ± 1.98
CBF	100.00 ± 0.00	99.87 ± 0.10	99.92 ± 0.05	99.99 ± 0.03
CinCECGT	83.36 ± 0.26	90.17 ± 1.05	90.51 ± 0.58	85.22 ± 0.33
ChlCon	81.43 ± 0.80	76.15 ± 0.73	79.80 ± 0.68	81.08 ± 0.49
Coffee	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00
Average	87.38 ± 0.74	85.18 ± 0.79	**88.61 ± 0.96**	**88.22 ± 0.59**
Ave. w/o Beef	87.83 ± 0.66	**88.97 ± 0.59**	**89.16 ± 0.96**	**88.65 ± 0.48**

Table 2
Training time comparison on pruning 50% of kernels of ROCKET-PPV-MAX. 'CV' shows the cross-validation time for finding optimal hyper-parameters, while 'Refit' shows the time cost for refitting a classifier with the decided hyper-parameters.

Dataset	ADMM-based (in Seconds)			POCKET (in Seconds)			
	CV	Refit	*Sum*	Stage 1 CV	Stage 1 Refit	Stage 2	*Sum*
Adiac	3306.55	199.65	*3506.20*	289.98	42.30	0.22	*332.50*
ArrowHead	2357.46	149.73	*2507.19*	252.06	20.01	0.02	*272.09*
Beef	2547.72	148.37	*2696.09*	310.08	19.92	0.02	*330.02*
BeetleFly	2451.91	148.23	*2600.14*	279.93	19.27	0.01	*299.21*
BirdChicken	2216.72	149.12	*2365.84*	249.44	18.73	0.01	*268.18*
Car	2650.60	152.21	*2802.81*	307.73	20.52	0.04	*328.29*
CBF	2295.02	153.04	*2448.06*	241.28	19.52	0.01	*260.81*
CinCECGT	2471.25	149.79	*2621.04*	293.84	19.82	0.02	*313.68*
ChlCon	3369.88	189.48	*3559.36*	248.16	20.38	0.31	*268.85*
Coffee	2411.54	152.27	*2563.81*	269.69	18.96	0.01	*288.66*
Average	2629.68	159.96	*2789.64*	**274.72**	**22.27**	0.07	***297.07***

Table 3
Validation training accuracy of our ADMM-based Algorithm on pruning ROCKET-PPV-MAX for the 'Beef' dataset. Here 'VT. Acc.' denotes 'Validation Training Accuracy'.

VT. Acc. (%) ρ_2 \ ρ_1	0.01	0.1	1	10	100
0.01	67.50	67.50	67.50	67.50	67.50
0.1	67.50	67.50	67.50	67.50	67.50
1	**87.50**	**87.50**	**87.50**	**87.50**	86.67
10	16.67	16.67	16.67	16.67	33.33
100	16.67	16.67	16.67	16.67	25.00

Table 4
Validation training accuracy of our POCKET on pruning ROCKET-PPV-MAX for the 'Beef' dataset.

k	0.01	0.1	1	10	100
VT. Acc. (%)	100.00	100.00	100.00	100.00	100.00

4.2. ADMM-based algorithm vs. POCKET

4.2.1. Settings

We first compare our ADMM-based algorithm with the accelerated POCKET on pruning 50% of kernels from ROCKET-PPV-MAX for the first 10 datasets of the UCR archive. To choose the optimal hyperparameters, which encompass ρ_1 and ρ_2 for the ADMM-based algorithm, and k for Stage 1 of POCKET, we use a 5-fold cross-validation process within the range of $[0.01, 0.1, 1, 10, 100]$, employing 25 threads for computational efficiency.

4.2.2. Results and discussions

Table 1 compares the test accuracy between the ADMM-based algorithm and POCKET. Notably, both algorithms can successfully remove redundancy without causing severe degradation in most cases. POCKET shows superior performance than the ADMM-based algorithm and even outperforms the original full model when 50% of kernels are pruned. We also compare the training time efficiency in Table 2, in which POCKET is approximately 10 times faster than the ADMM-based algorithm.

The 'Beef' dataset is an outlier, in which the ADMM-based algorithm performs more than 30% worse than POCKET, while they have similar accuracy on the other datasets. To explore the anomaly, we examine the validation test accuracy for both algorithms, which is consistently **0%** for various parameter values. This is due to the 'Beef' dataset's limited size, with only 30 training samples spread across 5 categories, resulting in each fold of the 5-fold cross-validation process containing only 1 training sample. In this scenario, where the values for hyperparameters—ρ_1 and ρ_2 for the ADMM-base algorithm, and k for POCKET—are chosen almost at random, the robustness of algorithms with respect to hyperparameters becomes a crucial factor.

We further investigate the training accuracy of both algorithms with different hyperparameters in Tables 3 and 4. It is shown that the validation training accuracy of the ADMM-based algorithm varies dramatically from 16.67% to 87.50% when ρ_1 and ρ_2 are changed, whereas those of POCKET remains consistently at 100% when k varies within the same range. This suggests that the ADMM-based algorithm is more sensitive to hyperparameters than POCKET, making it challenging to select appropriate values, especially when two hyperparameters (ρ_1 and ρ_2) are involved simultaneously.

For further verification, we also rerun the ADMM-based algorithm on the 'Beef' dataset, setting ρ_1 and ρ_2 to the values that produced the highest validation training accuracy in Table 3, namely 0.01 and 1, respectively. Remarkably, this adjustment results in a significant 30% improvement in test accuracy, indicating that the ADMM-based algorithm relies on more careful tuning of hyperparameters compared to POCKET.

As the ADMM-based algorithm is time-consuming and the core POCKET exhibits superior performance, the subsequent experiments will omit the former and concentrate solely on evaluating POCKET.

Table 5
Performance comparison (%) on pruning ROCKET-PPV-MAX.

Dataset	Original ROCKET	S-ROCKET [13]		Our POCKET		
	[11] Acc. ± Std.	Remain rate	Acc. ± Std.	Remain rate	Stage 1 Acc. ± Std.	Stage 2 Acc. ± Std.
Adiac	78.13 ± 0.43	100.00	78.13 ± 0.43	**50.00**	80.18 ± 0.54	79.87 ± 0.50
ArrowHead	81.37 ± 1.03	24.47	81.77 ± 1.31	24.47	80.86 ± 1.93	81.83 ± 1.59
Beef	82.00 ± 3.71	19.46	81.00 ± 2.60	19.46	82.67 ± 2.49	83.33 ± 3.65
BeetleFly	90.00 ± 0.00	21.08	90.00 ± 0.00	21.08	89.50 ± 1.50	90.00 ± 0.00
BirdChicken	90.00 ± 0.00	24.31	89.00 ± 3.00	24.31	90.00 ± 0.00	90.00 ± 0.00
Car	88.33 ± 1.83	34.29	88.33 ± 2.69	34.29	92.50 ± 0.83	91.67 ± 1.29
CBF	100.00 ± 0.00	17.98	99.96 ± 0.05	17.98	99.92 ± 0.07	99.98 ± 0.04
CinCECGT	83.61 ± 0.55	24.44	82.79 ± 0.74	24.44	90.86 ± 2.96	88.23 ± 1.64
ChlCon	81.50 ± 0.49	40.67	79.26 ± 1.46	40.67	79.43 ± 0.59	80.71 ± 0.60
Coffee	100.00 ± 0.00	18.06	100.00 ± 0.00	18.06	100.00 ± 0.00	100.00 ± 0.00
Computers	76.32 ± 0.84	26.75	76.80 ± 0.95	26.75	74.60 ± 1.82	77.20 ± 0.76
CricketX	81.92 ± 0.49	72.95	82.05 ± 0.62	72.95	82.10 ± 0.66	82.18 ± 0.80
CricketY	85.38 ± 0.60	57.50	85.08 ± 0.56	52.50	83.87 ± 0.75	84.90 ± 0.71
CricketZ	85.44 ± 0.64	70.40	85.03 ± 0.69	70.40	83.87 ± 0.71	85.10 ± 0.71
DiaSizRed	97.09 ± 0.61	24.23	96.50 ± 0.84	24.23	95.59 ± 2.37	97.84 ± 0.55
DisPhaOAG	75.68 ± 0.63	32.85	74.89 ± 0.99	32.85	74.89 ± 0.68	73.74 ± 0.98
DisPhaOutCor	76.74 ± 0.88	32.71	77.03 ± 1.30	32.71	77.54 ± 0.96	76.27 ± 1.61
DoLoDay	60.50 ± 1.70	55.20	60.25 ± 1.84	50.20	60.88 ± 1.68	60.62 ± 2.11
DoLoGam	86.09 ± 0.54	18.43	86.45 ± 0.80	18.43	88.33 ± 0.68	88.26 ± 0.63
DoLoWKE	97.68 ± 0.29	18.28	97.54 ± 0.35	18.28	98.55 ± 0.00	98.48 ± 0.22
Earthquakes	74.82 ± 0.00	32.64	74.82 ± 0.00	32.64	74.96 ± 0.29	74.96 ± 0.54
ECG200	90.40 ± 0.49	10.88	89.90 ± 0.70	10.88	91.20 ± 1.17	90.60 ± 0.66
ECG5000	94.75 ± 0.05	33.91	94.68 ± 0.09	33.91	93.55 ± 0.60	94.78 ± 0.06
ECGFiveDays	100.00 ± 0.00	22.19	100.00 ± 0.00	22.19	100.00 ± 0.00	100.00 ± 0.00
EOGHSignal	58.26 ± 1.05	69.33	58.01 ± 1.19	64.33	59.17 ± 1.46	58.45 ± 1.18
EOGVSignal	54.70 ± 0.58	51.56	55.03 ± 0.91	51.56	55.28 ± 0.94	54.81 ± 0.58
FaceAll	94.68 ± 0.40	46.61	94.64 ± 0.32	46.61	94.16 ± 0.67	94.14 ± 0.78
FaceFour	97.61 ± 0.34	18.30	97.61 ± 0.34	18.30	98.41 ± 0.56	97.84 ± 0.34
FacesUCR	96.20 ± 0.09	48.43	96.20 ± 0.08	48.43	96.14 ± 0.16	96.34 ± 0.08
FiftyWords	82.99 ± 0.41	100.00	82.99 ± 0.41	50.00	82.00 ± 0.47	82.46 ± 0.47
AVERAGE	84.74 ± 0.62	38.93	84.52 ± 0.84	**35.10**	**85.03 ± 0.92**	**85.15 ± 0.77**

Table 6
Comparison of average training time (for one run) in different steps or stages for ROCKET-PPV-MAX. 'Stage 1 CV' shows the cross-validation time for finding optimal hyper-parameter k in Stage 1, while 'Stage 1 Refit' shows the time cost for refitting the classifier with the decided k. '*Sum*' shows the total training time.

Dataset	S-ROCKET [13] (in Seconds)				POCKET (in Seconds)			
	Pre-train	Kernel Selection	Post training	*Sum*	Stage 1 CV	Stage 1 Refit	Stage 2	*Sum*
Adiac	0.39	5550.66	0.39	*5551.44*	226.68	43.79	0.22	*270.69*
ArrowHead	0.02	398.64	0.02	*398.68*	117.28	19.03	0.02	*136.33*
Beef	0.02	411.45	0.02	*411.49*	114.50	19.39	0.01	*133.90*
BeetleFly	0.01	200.08	0.01	*200.10*	102.49	18.02	0.01	*120.52*
BirdChicken	0.01	199.94	0.01	*199.96*	103.79	18.01	0.01	*121.81*
Car	0.04	699.04	0.06	*699.14*	107.39	19.62	0.05	*127.06*
CBF	0.02	344.29	0.02	*344.33*	118.49	18.90	0.01	*137.40*
CinCECGT	0.02	482.37	0.03	*482.42*	113.93	19.14	0.01	*133.08*
ChlCon	0.44	2228.06	0.37	*2228.87*	141.75	20.92	0.24	*162.91*
Coffee	0.02	269.04	0.02	*269.08*	106.25	18.61	0.01	*124.87*
Computers	0.20	1808.80	0.18	*1809.18*	115.01	18.88	0.09	*133.98*
CricketX	0.37	2821.16	0.34	*2821.87*	141.94	24.92	0.31	*167.17*
CricketY	0.37	2834.85	0.33	*2835.55*	140.86	24.94	0.25	*166.05*
CricketZ	0.36	2836.02	0.32	*2836.70*	143.78	25.30	0.28	*169.36*
DiaSizRed	0.02	209.85	0.01	*209.88*	105.53	19.35	0.02	*124.90*
DisPhaOAG	0.37	1898.41	0.34	*1899.12*	126.13	20.26	0.16	*146.55*
DisPhaOutCor	0.58	2498.19	0.50	*2499.27*	126.50	19.78	0.29	*146.57*
DoLoDay	0.04	512.63	0.07	*512.74*	128.89	25.26	0.07	*154.22*
DoLoGam	0.01	199.82	0.02	*199.85*	105.36	18.12	0.02	*123.50*
DoLoWKE	0.01	199.69	0.02	*199.72*	104.61	17.20	0.01	*121.82*
Earthquakes	0.29	2309.31	0.26	*2309.86*	115.42	19.46	0.14	*135.02*
ECG200	0.06	784.47	0.08	*784.61*	116.22	19.54	0.05	*135.81*
ECG5000	0.48	2686.28	0.42	*2687.18*	152.17	20.57	0.26	*173.00*
ECGFiveDays	0.01	205.93	0.02	*205.96*	106.90	17.91	0.01	*124.82*
EOGHSignal	0.35	2626.89	0.29	*2627.53*	143.25	25.34	0.24	*168.83*
EOGVSignal	0.34	2634.98	0.29	*2635.61*	139.29	25.59	0.20	*165.08*
FaceAll	0.53	4372.66	0.46	*4373.65*	164.16	31.94	0.37	*196.47*
FaceFour	0.01	321.44	0.02	*321.47*	106.71	19.10	0.01	*125.82*
FacesUCR	0.12	1599.16	0.12	*1599.40*	151.37	31.75	0.08	*183.20*
FiftyWords	0.44	8067.68	0.41	*8068.53*	264.88	43.44	0.30	*308.62*
AVERAGE	0.20	1740.39	0.18	*1740.77*	131.72	22.80	0.13	***154.65***

Table 7
Performance comparison (%) on pruning ROCKET-PPV.

Dataset	Original ROCKET [11] Acc.[a]	Remain Rate[a]	Random Acc.[a]	ℓ_1-norm [14] Acc.[a]	Soft Filter [65] Acc.[a]	S-ROCKET [13] Acc.[a]	Our POCKET (%)		
							Original ROCKET [11] Acc. ± Std.	Stage 1 Acc. ± Std.	Stage 2 Acc. ± Std.
Adiac	78	41	69	72	71	78	78.34 ± 0.64	80.20 ± 0.57	79.67 ± 0.50
ArrowHead	83	41	79	82	82	83	83.43 ± 2.09	85.03 ± 1.30	83.49 ± 1.24
Beef	82	16	70	80	80	82	82.33 ± 1.53	82.33 ± 2.13	82.67 ± 1.33
BeetleFly	95	27	86	91	91	95	95.00 ± 0.00	96.00 ± 2.00	95.00 ± 0.00
BirdChicken	90	20	78	83	83	88	90.00 ± 0.00	90.00 ± 0.00	90.00 ± 0.00
Car	90	19	81	85	84	87	88.00 ± 5.15	92.33 ± 1.33	93.17 ± 1.38
CBF	100	19	85	97	99	100	99.89 ± 0.00	99.89 ± 0.00	99.89 ± 0.00
CinCECGT	81	21	71	80	81	80	80.48 ± 0.88	83.17 ± 1.93	82.51 ± 1.20
ChlCon	76	31	62	73	73	73	76.26 ± 0.54	73.82 ± 1.21	75.55 ± 0.65
Coffee	100	58	95	98	98	99	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00
Computers	77	29	69	76	74	77	77.32 ± 0.51	78.36 ± 0.63	77.96 ± 0.85
CricketX	83	72	81	82	82	83	82.82 ± 0.79	82.46 ± 0.74	82.95 ± 0.79
CricketY	85	71	78	80	80	85	84.74 ± 0.80	84.10 ± 1.12	84.79 ± 0.90
CricketZ	85	70	79	81	82	85	84.87 ± 0.57	84.38 ± 0.79	84.79 ± 0.59
DiaSizRed	95	29	88	94	94	95	95.36 ± 0.70	96.34 ± 0.91	95.52 ± 1.15
DisPhaOAG	75	35	69	73	73	75	74.89 ± 0.88	74.24 ± 1.36	74.82 ± 1.47
DisPhaOutCor	77	35	69	74	75	77	76.96 ± 1.08	76.96 ± 1.39	77.39 ± 1.60
DoLoDay	65	69	58	54	56	64	61.75 ± 1.87	62.12 ± 1.77	62.12 ± 2.17
DoLoGam	80	23	75	78	80	81	84.42 ± 0.87	82.83 ± 0.80	83.91 ± 0.78
DoLoWKE	97	1	82	92	92	94	97.83 ± 0.00	98.12 ± 0.35	98.12 ± 0.35
Earthquakes	75	1	69	75	75	75	74.82 ± 0.00	75.47 ± 1.04	75.32 ± 0.97
ECG200	90	18	81	89	89	88	89.80 ± 0.75	88.80 ± 1.60	89.80 ± 0.75
ECG5000	95	29	73	89	89	95	94.67 ± 0.05	94.56 ± 0.06	94.63 ± 0.05
ECGFiveDays	100	21	89	98	98	100	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00
EOGHSignal	57	56	49	55	55	55	57.07 ± 0.98	55.58 ± 1.31	56.71 ± 0.95
EOGVSignal	44	59	41	41	42	44	44.97 ± 1.13	44.39 ± 1.70	44.78 ± 1.34
FaceAll	79	52	69	78	78	80	79.46 ± 0.36	79.59 ± 0.37	79.50 ± 0.41
FaceFour	100	59	82	98	99	100	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00
FacesUCR	96	50	88	92	91	96	96.30 ± 0.13	96.10 ± 0.23	96.30 ± 0.15
FiftyWords	85	78	76	66	65	85	84.79 ± 0.35	85.10 ± 0.55	84.95 ± 0.46
AVERAGE	84	39	75	80	80	83	83.89 ± 0.76	**84.08 ± 0.91**	**84.21 ± 0.73**

[a] The data columns are taken from S-ROCKET [13], while those without it are achieved by reproduction under the same settings.

4.3. Comparing with the current state of the art

4.3.1. Settings

In this section, we compare our POCKET with the state-of-the-art pruning method for random kernels, S-ROCKET [13]. Following S-ROCKET, we prune the models for the first 30 datasets of the UCR archive. Referring to ROCKET and S-ROCKET, here and in the rest of the experiments, we extend the range of values for k in Stage 1 of POCKET for validation to $[0.01, 0.1, 1, 10, 100, 1000]$ for a comprehensive investigation. All threads are used to efficiently evaluate POCKET and the compared methods.

As S-ROCKET is very time-consuming, when comparing with S-ROCKET on pruning ROCKET-PPV and MINIROCKET, we directly present the results reported in [13]. For a fair comparison, we set the pruning rates and retraining settings in Stage 2 of our POCKET the same as S-ROCKET.

To offer a more comprehensive evaluation, we also compare POCKET and S-ROCKET on pruning ROCKET-PPV-MAX. Since S-ROCKET had not pruned ROCKET-PPV-MAX, we use its official code and reported settings for reproduction. For a fair comparison, POCKET adopts the pruning rates determined by S-ROCKET. However, when S-ROCKET produces an invalid pruning rate, i.e., when 100% of the kernels remain, we eliminate 50% of the random kernels.

4.3.2. Results and discussions

The results of pruning ROCKET-PPV-MAX and the training time comparison for different steps or stages between S-ROCKET and POCKET are presented in Tables 5 and 6, respectively. Tables 7 and 8 compare the pruned accuracy of ROCKET-PPV and MINIROCKET. For comprehensive evaluation, we also report the overall results on pruning ROCKET-PPV-MAX across all 85 'bake off' and 43 'extra' UCR datasets in Tables 9 and 10 in Appendices A and B, respectively.

On average, POCKET outperforms its counterparts by 0.5% to 4% in pruned accuracy across the three models, while being 11× faster than S-ROCKET. Specifically, despite removing more than 60% of random kernels in ROCKET-PPV-MAX and ROCKET-PPV, neither Stage 1 nor Stage 2 of POCKET suffers from severe degradation; they even achieve higher average accuracy than the original unpruned models. The ability of POCKET to reduce redundant features effectively decreases models' complexity, thereby alleviating the overfitting issue and resulting in improved accuracy. This improvement is also observed in deep learning compression [15,53], and we will further verify it in the next Section 4.4.

As for MINIROCKET, since its capacity is limited by the use of discrete binary kernels, pruning is more likely to cause significant degradation. Consequently, the pruned accuracy of MINIROCKET achieved in Stage 1 of POCKET decreases by 0.89%. However, Stage 2, which involves ℓ_2-regularized refitting, significantly enhances POCKET's performance, reducing the gap between the original and pruned MINIROCKET to just 0.25%.

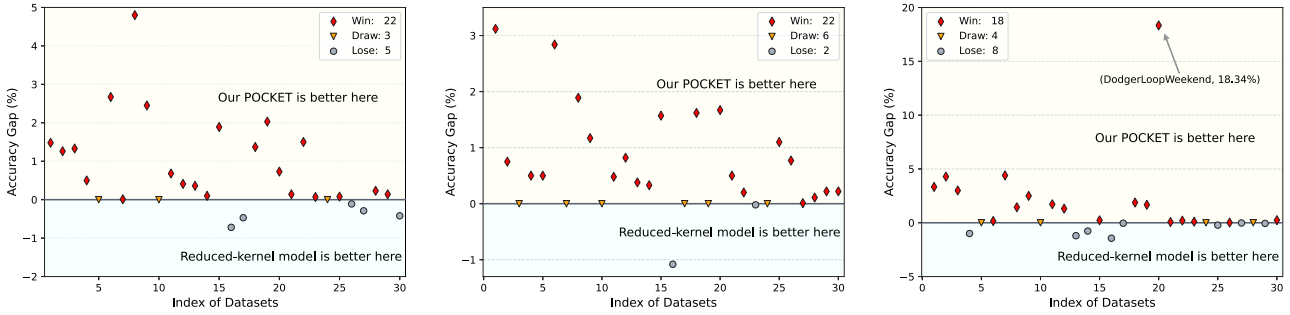
In Table 5, while S-ROCKET [13] suffers from an invalid pruning rate in the 'Adiac' dataset, our POCKET eliminates redundancy with a pre-set rate, showing the notable advantage of POCKET in enabling customized pruning rates. Additionally, although the MAX feature has been considered less important than the PPV [12,13], it still contributes to a 1% accuracy improvement for ROCKET-PPV-MAX after pruning, compared to ROCKET-PPV. This suggests that the importance of MAX may be underestimated.

Comparing with S-ROCKET, the efficiency of Stage 1 enables POCKET to operate 11× faster. However, Stage 1 costs a significant amount of time in cross-validation to determine the optimal value for k, indicating the necessity for more efficient parameter selection techniques.

Table 8
Performance comparison (%) on pruning MINIROCKET.

Dataset	Original MINIROCKET Acc. [12][a]	Remain Rate[a]	Random Acc.[a]	ℓ_1-norm [14] Acc.[a]	Soft Filter [65] Acc.[a]	S-ROCKET [13] Acc.[a]	Our POCKET		
							Original MINIROCKET [12] Acc. ± Std.	Stage 1 Acc. ± Std.	Stage 2 Acc. ± Std.
Adiac	82	10	63	71	72	78	82.12 ± 0.53	81.10 ± 0.82	81.61 ± 0.90
ArrowHead	87	35	81	83	83	86	86.51 ± 0.64	88.74 ± 1.14	87.20 ± 0.89
Beef	87	10	73	79	81	83	86.67 ± 0.00	84.00 ± 3.59	86.67 ± 1.49
BeetleFly	88	30	79	86	86	87	89.00 ± 2.00	85.50 ± 1.50	89.50 ± 1.50
BirdChicken	90	19	81	84	86	89	90.00 ± 0.00	90.00 ± 0.00	90.00 ± 0.00
Car	92	49	78	90	91	92	91.67 ± 0.00	91.67 ± 0.00	91.67 ± 0.00
CBF	100	1	89	96	98	100	99.89 ± 0.00	98.20 ± 0.79	99.67 ± 0.26
CinCECGT	87	39	65	75	76	84	86.33 ± 0.32	86.64 ± 0.94	86.83 ± 0.52
ChlCon	76	30	53	72	72	72	76.02 ± 0.49	73.90 ± 0.60	75.39 ± 0.36
Coffee	100	33	87	99	99	100	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00
Computers	72	34	64	71	69	73	72.88 ± 0.64	71.92 ± 0.96	72.84 ± 1.33
CricketX	82	19	66	71	73	79	81.10 ± 0.57	79.23 ± 1.31	80.26 ± 0.79
CricketY	83	20	76	80	79	81	83.08 ± 0.53	78.56 ± 1.31	80.56 ± 0.96
CricketZ	83	62	73	80	81	82	82.62 ± 0.32	82.54 ± 0.57	82.74 ± 0.44
DiaSizRed	93	54	85	90	90	93	92.65 ± 0.16	93.30 ± 0.37	92.84 ± 0.18
DisPhaOAG	75	27	57	71	72	75	74.32 ± 0.72	73.74 ± 1.45	74.39 ± 1.33
DisPhaOutCor	78	33	69	76	76	78	77.39 ± 1.01	76.09 ± 1.52	76.23 ± 1.15
DoLoDay	59	73	34	51	53	59	62.12 ± 1.68	61.75 ± 1.15	62.75 ± 1.66
DoLoGam	84	40	68	80	80	84	84.20 ± 0.54	83.55 ± 0.65	83.48 ± 0.71
DoLoWKE	98	1	90	95	95	97	98.55 ± 0.00	98.55 ± 0.00	98.55 ± 0.00
Earthquakes	75	1	59	73	74	75	74.82 ± 0.00	74.82 ± 0.00	75.25 ± 1.03
ECG200	92	20	84	89	90	91	91.70 ± 0.46	91.00 ± 1.10	91.10 ± 0.70
ECG5000	94	39	81	93	93	39	94.49 ± 0.05	94.44 ± 0.09	94.38 ± 0.06
ECGFiveDays	100	21	86	97	98	100	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00
EOGHSignal	60	29	46	56	57	59	59.78 ± 1.12	56.19 ± 1.38	58.15 ± 1.18
EOGVSignal	54	37	41	48	47	52	53.90 ± 1.26	50.28 ± 2.33	52.07 ± 1.75
FaceAll	81	72	71	79	79	81	80.65 ± 0.17	80.88 ± 0.17	80.79 ± 0.12
FaceFour	99	67	84	96	96	99	98.86 ± 0.00	98.52 ± 0.52	98.86 ± 0.00
FacesUCR	96	80	86	92	91	96	95.78 ± 0.18	95.73 ± 0.14	95.75 ± 0.17
FiftyWords	84	78	77	81	81	84	84.11 ± 0.42	83.60 ± 0.53	84.15 ± 0.44
AVERAGE	84	36	72	80	81	83	84.37 ± 0.46	**83.48 ± 0.83**	**84.12 ± 0.66**

[a] The columns are taken from S-ROCKET [13].



(a) ROCKET-PPV-MAX (b) ROCKET-PPV (c) MINIROCKET

Fig. 2. Models pruned by POCKET vs. Reduced-kernel models trained from scratch.

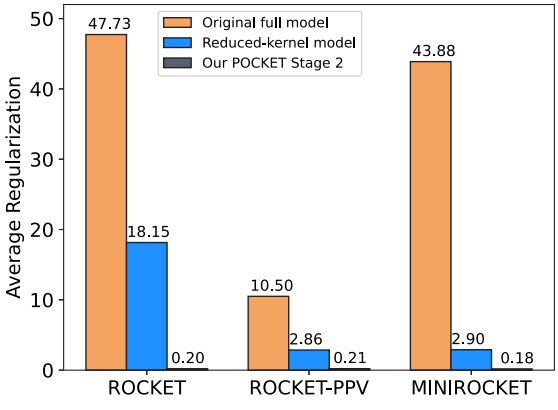


Fig. 3. Average ℓ_2 regularization strength of models across datasets involved in Fig. 2.

4.4. Effectiveness verification

4.4.1. Settings

We verify the efficacy of POCKET for pruning by comparing its pruned models with '**reduced-kernel models**', which refers to models trained from scratch with the number of kernels reduced to match the level achieved by POCKET in the initialization phase, rather than the default 10K. The other settings for reduced-kernel models are the same as the original full models. Without loss of generality, we directly adopt the pruning rates derived from S-ROCKET as outlined in Tables 5, 7 and 8 for the ROCKET-PPV-MAX, ROCKET-PPV and MINIROCKET models, respectively.

4.4.2. Results and discussions

As illustrated in Fig. 2, POCKET consistently outperforms the reduced kernel models in most datasets, showing the efficacy of POCKET in identifying crucial features to enhance performance.

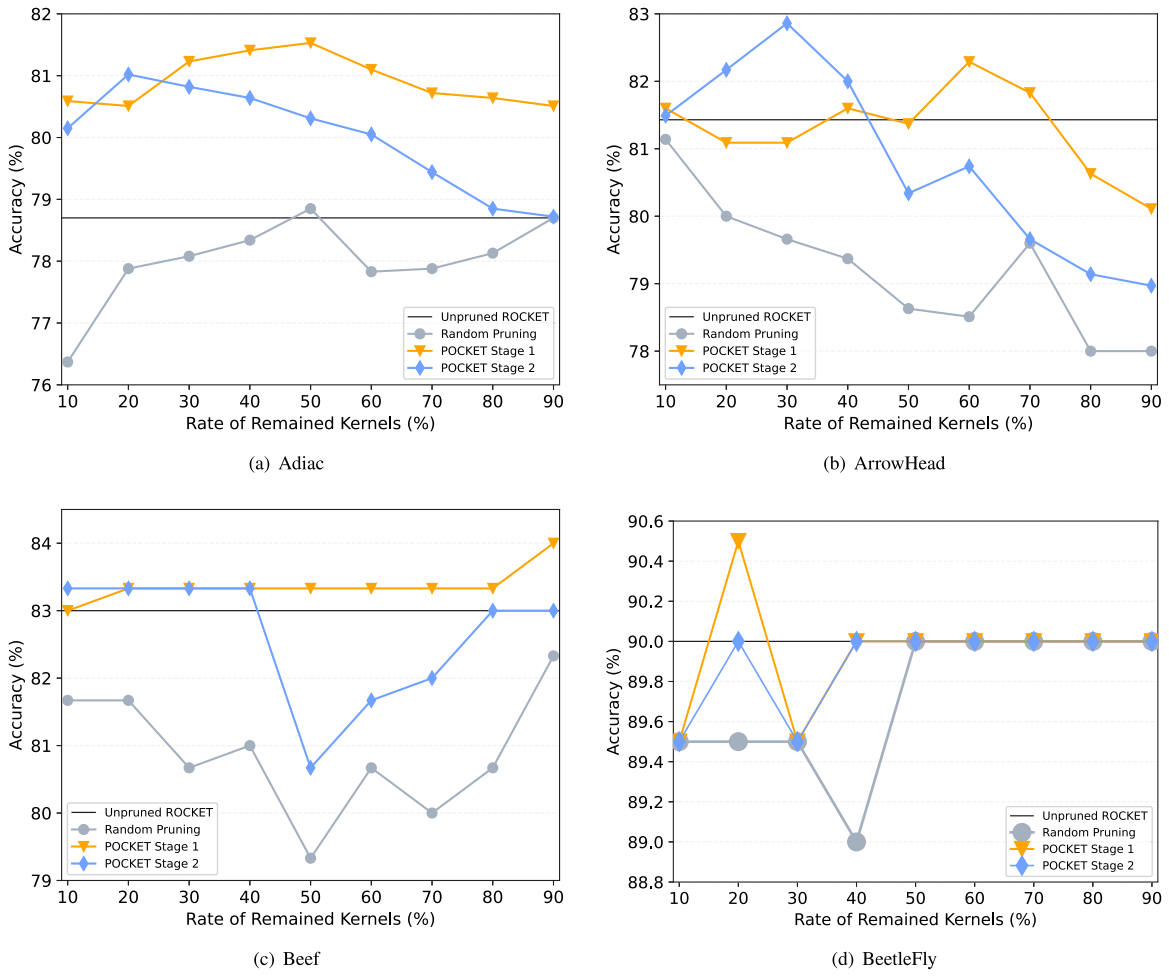


Fig. 4. Pruning ROCKET-PPV-MAX for the first four datasets under different pruning rates.

In Fig. 3, we visualize the average regularization strength (ρ_1) of models across datasets. The strengths automatically selected by the efficient Leave-One-Out validation indicate the degree of redundancy. Not surprisingly, the original models (with full 10K kernels) have the strongest penalty strengths, followed by the reduced-kernel model. The models pruned by our POCKET suffer the lightest penalty strength, suggesting that the remaining random kernels and features are crucial, while those pruned by POCKET are indeed redundant. This observation also explains why models pruned by POCKET sometimes outperform the full models—the overfitting issue is mitigated by eliminating redundant parameters.

4.5. Pruning rate

4.5.1. Settings

An advantage of POCKET is its adjustable pruning rate, which can be tailored to the computational budget. To evaluate POCKET under different pruning rates, we prune 10% to 90% of random kernels on ROCKET-PPV-MAX and evaluate different stages of POCKET across the first four datasets of the UCR archive. In comparison, we also assess Random Pruning under identical settings. Random Pruning removes kernels randomly, providing a baseline for analyzing the impact of pruning rates.

4.5.2. Results and discussions

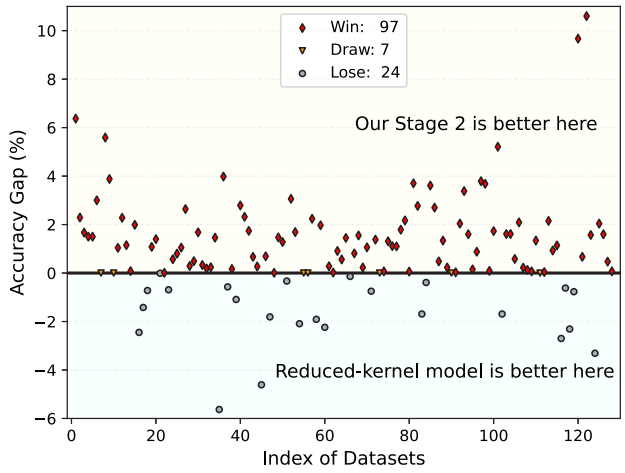
The results are illustrated in Fig. 4. Generally, pruning kernels will cause degradation in accuracy, since the expression capacity of the model is reduced. However, as POCKET is efficient in recognizing redundancy, a relatively obvious improvement in accuracy, empirically a maximum of 3%, can be achieved when 10% to 30% of kernels are removed. Both Stage 1 and Stage 2 of POCKET exhibit superior performance compared to random pruning with a substantial margin across different pruning rates and datasets. Notably, in the 'Adiac' dataset, POCKET outperforms the unpruned models at each pruning rate, underscoring its effectiveness in eliminating redundant features.

As discussed in 4.3.2, the optional Stage 2 is less advantageous for ROCKET-PPV-MAX than for MINIROCKET in further enhancing accuracy. In the 'Beef' dataset, Stage 2 exhibits similar trends to random pruning. Both Stage 2 and Random Pruning employ the same efficient Leave-one-Out validation approach, using the mean squared error as the evaluation metric instead of classification accuracy. From the perspective of classification accuracy, the selected values for hyperparameters may not be optimal, resulting in poor performance. A more refined hyperparameter selection space and strategy could potentially ameliorate the situation, but would consume more computation time.

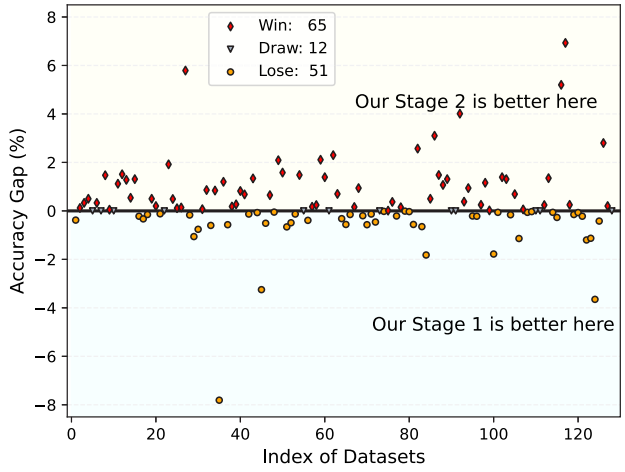
4.6. More datasets

4.6.1. Settings

We proceed to comprehensively evaluate POCKET by pruning ROCKET-PPV across all 128 datasets of the UCR archive. We keep only 10% of random kernels for each dataset. Note that in the 'Fungi' dataset, each class contains only one training sample, making cross-validation unavailable. To address this, we directly set the hyperparameter $k = 1$. All other experimental settings remain consistent with those detailed in Section 4.1.3.



(a) POCKET Stage 2 vs. reduced-kernel models from scratch



(b) POCKET Stage 2 vs. POCKET Stage 1

Fig. 5. Accuracy comparison across 128 UCR datasets when 90% of kernels of ROCKET-PPV are pruned by POCKET.

4.6.2. Results and discussions

Fig. 5-(a) and (b) present comparisons between Stage 2 and reduced-kernel models, as well as Stage 1, respectively. Across the majority of datasets (97 out of 128), POCKET demonstrates superior performance over the reduced-kernel ROCKET models.

When comparing Stage 1 with Stage 2, the latter exhibits a slight performance improvement by a narrow margin on more datasets. In several specific instances (datasets indexed 35, 45, and 124), the reduced-kernel ROCKET models outperform Stage 2, while Stage 1 narrows the performance gaps. This aligns with the findings presented in the previous Section 4.5, where a non-optimal small penalty value selected through Leave-One-Out validation exacerbates the overfitting issue. The nuanced performance dynamics between Stage 1 and Stage 2 underscore the importance of fine-tuning hyperparameters to strike the right balance between model complexity and overfitting across diverse datasets.

4.7. Hyperparameter k

4.7.1. Settings

To explore the impact of the hyperparameter k on Stage 1 of POCKET, we visualize the average cross-validation training and testing accuracy over ten repetitions on 127 datasets, excluding the 'Fungi' dataset, whose k is directly set to 1. The experimental setup aligns with the parameters outlined in the previous experiment in 4.6.

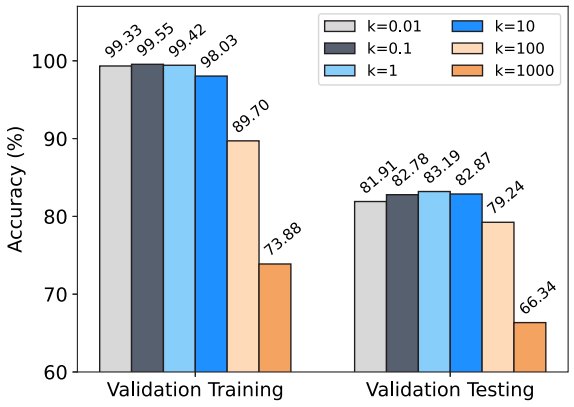


Fig. 6. Average cross-validation accuracy of different k over 127 datasets.

4.7.2. Results and discussions

The results presented in Fig. 6 indicate that the set {0.1, 1, 10} serves an appropriate range to achieve relatively higher cross-validation accuracy for both training and testing. Further refinement within the span of [0.1, 10] has the potential to enhance the performance of POCKET. In addition, the value $k = 1$ is an appropriate choice for direct application without cross-validation. Oppositely, the value $k = 1000$ is too large, thereby compromising classification accuracy across the majority of datasets. Since a small k prioritizes validation accuracy, while a large k emphasizes group sparsity, the excessively large $k = 1000$ results in performance degradation dramatically.

4.8. Convergence of Stage 1

4.8.1. Settings

We visualize the convergence process of POCKET Stage 1 across the first nine UCR datasets. The experimental settings are the same as in experiment 4.6, where 90% of random kernels on ROCKET-PPV are pruned, except that the number of iterations is extended to 200 to comprehensively show the convergence process. To avoid smoothing of convergence characteristics, we execute a single run for each dataset, rather than the ten repetitions.

4.8.2. Results and discussions

In Fig. 7, we plot the norm gap between $\mathbf{W}$ and $\boldsymbol{\Theta}$, i.e., $\|\mathbf{W} - \boldsymbol{\Theta}\|_2$. We also present the relative threshold, defined as $\frac{\|\widetilde{\mathbf{W}}'^{(t)}_{f_1:}\|_2}{\|\widetilde{\mathbf{W}}'^{(t)}_{f_{m+1}:}\|_2}$, where $\|\widetilde{\mathbf{W}}'^{(t)}_{f_1:}\|_2$ and $\|\widetilde{\mathbf{W}}'^{(t)}_{f_{m+1}:}\|_2$ are defined in (17). Using the relative threshold, as opposed to the direct threshold, mitigates the impact of the overall magnitude of the weights during iterations, providing a clearer illustration of the convergence.

It is shown that with an increasing number of iterations, the relative threshold progressively decreases towards 0, indicating a rapid convergence of $\mathbf{W}$ towards $\boldsymbol{\Theta}$. Stage 1 of POCKET achieves convergence in 50 to 200 iterations across most datasets, with notable instances such as 'ArrowHead', 'Beef', and 'BirdChicken' where convergence is achieved in just 30 iterations. Considering the differing convergence epochs observed in various datasets, adaptive stopping criteria could potentially further expedite POCKET. For example, instead of fixing T, terminating the iteration when the norm gap falls below a predefined threshold could be more effective.

5. Conclusion

This paper introduces two algorithms, the ADMM-based algorithm and the core POCKET, to efficiently prune models of the ROCKET

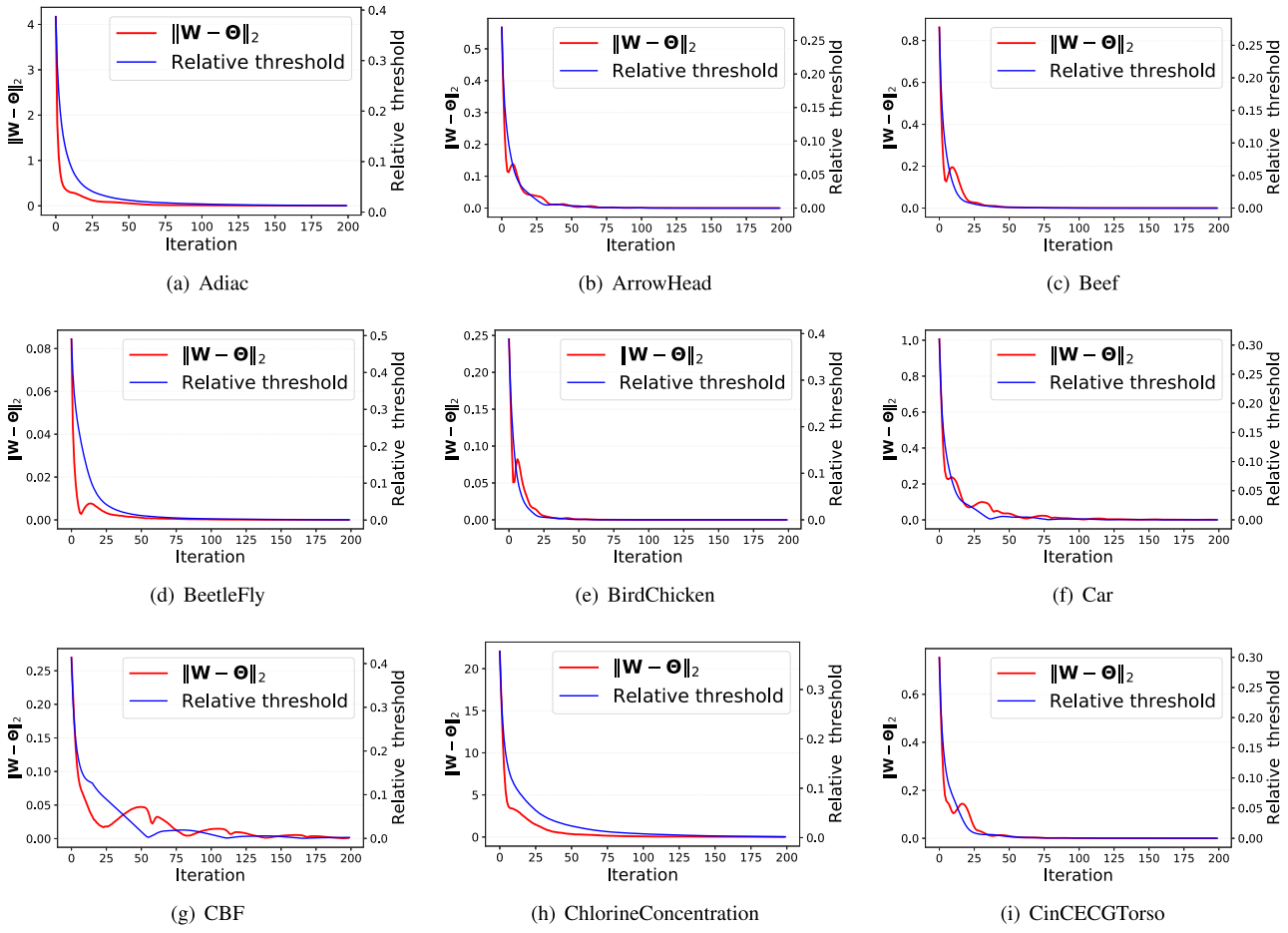


Fig. 7. Convergence process of Stage 1 in POCKET on pruning ROCKET-PPV. Here 90% of the kernels are pruned.

family from a feature selection perspective, thus avoiding the computational inefficiency caused by the direct evaluation of random kernels. Based on the ADMM-based algorithm, POCKET employs a two-stage strategy that significantly accelerates the pruning process, resulting in substantially faster speeds and higher accuracy compared with existing pruning methods. Experimental results demonstrate that POCKET is effective in identifying redundant kernels and features. Specifically, on ROCKET-PPV-MAX, POCKET achieves an average kernel reduction exceeding 60% without accuracy degradation, but improvements instead.

In future work, we intend to integrate the two classifiers generated during the two stages of POCKET for further enhancement in performance, investigate more streamlined cross-validation techniques for higher efficiency, and explore the application of POCKET beyond time series classification.

CRediT authorship contribution statement

Shaowu Chen: Writing – original draft, Visualization, Software, Methodology, Conceptualization. **Weize Sun:** Writing – review & editing, Resources, Project administration, Methodology, Funding acquisition, Formal analysis. **Lei Huang:** Writing – review & editing, Supervision, Resources, Methodology, Funding acquisition. **Xiao Peng Li:** Writing – review & editing, Resources, Conceptualization. **Qingyuan Wang:** Writing – review & editing, Visualization, Software. **Deepu John:** Writing – review & editing, Resources, Conceptualization.

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.

Data availability

We have shared a link to my code in the manuscript.

Acknowledgments

This work was supported in part by the Guangdong Basic and Applied Basic Research Foundation, China under Grant 2021A1515011706, the National Science Fund for Distinguished Young Scholars, China under Grant 61925108, the Key Project of International Cooperation and Exchanges of the National Natural Science Foundation of China under Grant 62220106009, the project of Shenzhen Peacock Plan Teams, China under Grant KQTD20210811090051046, the Open Research Fund from the Guangdong Provincial Key Laboratory of Big Data Computing, the Chinese University of Hong Kong, Shenzhen, China, under Grant B10120210117-OF07, the Shenzhen University 2035 Program for Excellent Research, China under Grant 2022B009, the Research Team Cultivation Program of Shenzhen University, China under Grant 2023DFT003, and the Microelectronic Circuits Center Ireland under Grant MCCI-2020-07.

We extend our gratitude to Mr. Angus Dempster and Dr. Hojjat Salehinejad for their shared code and prompt email responses. We acknowledge the assistance of OpenAI's language model, ChatGPT, in proofreading and enhancing the clarity of this manuscript. However, it is important to note that the generation of content lies solely with the authors.

Appendix A. 85 'Bake off' UCR datasets

See Table 9.

Table 9
Performance comparison on pruning ROCKET-PPV-MAX across 85 'Bake off' UCR datasets. 'Acc.' stands for 'Accuracy'; '***AVERAGE***' represents the average over all listed datasets.

Dataset	r-STSF [29]	L-Time [48]	H-ITime [47]	ROCKET & Pruning							
				Unpruned [11]	Pruned models' Acc. (%)			#Remaining kernels		Overall pruning time (s)	
	Acc.[a] (%)	Acc.[a] (%)	Acc.[a] (%)	ROCKET Acc.	S-ROCKET [13]	POCKET Stage 1	POCKET Stage 2	S-ROCKET [13]	POCKET	S-ROCKET [13]	POCKET
AVERAGE	84.24	84.46	84.82	85.04 ± 0.51	84.88 ± 0.73	**85.17 ± 0.84**	85.05 ± 0.72	3861.64	**3226.35**	3433.97	**222.95**
Adiac	83.55	83.38	84.40	78.13 ± 0.43	78.13 ± 0.43	80.18 ± 0.54	79.87 ± 0.50	10 000.00	5000.00	5551.44	270.69
ArrowHead	74.06	84.00	89.14	81.37 ± 1.03	81.77 ± 1.31	80.86 ± 1.93	81.83 ± 1.59	2447.30	2447.30	398.68	136.33
Beef	87.00	76.67	70.00	82.00 ± 3.71	81.00 ± 2.60	82.67 ± 2.49	83.33 ± 3.65	1945.80	1945.80	411.49	133.90
BeetleFly	92.00	90.00	75.00	90.00 ± 0.00	90.00 ± 0.00	89.50 ± 1.50	90.00 ± 0.00	2107.80	2107.80	200.10	120.52
BirdChicken	90.50	90.00	95.00	90.00 ± 0.00	89.00 ± 3.00	90.00 ± 0.00	90.00 ± 0.00	2430.50	2430.50	199.96	121.81
Car	87.50	93.33	90.00	88.33 ± 1.83	88.33 ± 2.69	92.50 ± 0.83	91.67 ± 1.29	3428.50	3428.50	699.14	127.06
CBF	99.17	99.89	99.78	100.00 ± 0.00	99.96 ± 0.05	99.92 ± 0.07	99.98 ± 0.04	1797.90	1797.90	344.33	137.40
ChlCon	77.82	84.84	88.26	81.50 ± 0.49	79.26 ± 1.46	79.43 ± 0.59	80.71 ± 0.60	4066.90	4066.90	2228.87	162.91
CinCECGTorso	99.93	87.32	84.42	83.61 ± 0.55	82.79 ± 0.74	90.86 ± 2.96	88.23 ± 1.64	2443.60	2443.60	482.42	133.08
Coffee	100.00	100.00	100.00	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	1805.60	1805.60	269.08	124.87
Computers	73.20	81.20	81.20	76.32 ± 0.84	76.80 ± 0.95	74.60 ± 1.82	77.20 ± 0.76	2674.60	2674.60	1809.18	133.98
CricketX	74.51	83.08	85.38	81.92 ± 0.49	82.05 ± 0.62	82.10 ± 0.66	82.18 ± 0.80	7294.70	7294.70	2821.87	167.17
CricketY	77.21	87.69	85.13	85.38 ± 0.60	85.08 ± 0.56	83.87 ± 0.75	84.90 ± 0.71	5750.10	5250.10	2835.55	166.05
CricketZ	77.23	86.15	85.38	85.44 ± 0.64	85.03 ± 0.69	83.87 ± 0.71	85.10 ± 0.71	7040.30	7040.30	2836.70	169.36
DiaSizRed	92.52	96.08	94.77	97.09 ± 0.61	96.50 ± 0.84	95.59 ± 2.37	97.84 ± 0.55	2423.20	2423.20	209.88	124.90
DisPhaOutAG	73.02	71.22	74.10	75.68 ± 0.63	74.89 ± 0.99	74.89 ± 0.68	73.74 ± 0.98	3284.80	3284.80	1899.12	146.55
DisPhaOutCor	78.12	76.09	79.71	76.74 ± 0.88	77.03 ± 1.30	77.54 ± 0.96	76.27 ± 1.61	3271.00	3271.00	2499.27	146.57
DisPhaTW[b]	68.06	71.22	65.47	71.94 ± 0.00	70.86 ± 1.17	70.29 ± 1.82	68.85 ± 2.14	2389.40	2389.40	2013.88	123.28
Earthquakes	75.25	73.38	72.66	74.82 ± 0.00	74.82 ± 0.00	74.96 ± 0.29	74.96 ± 0.54	3263.70	3263.70	2309.86	135.02
ECG200	89.70	92.00	90.00	90.40 ± 0.49	89.90 ± 0.70	91.20 ± 1.17	90.60 ± 0.66	1088.10	1088.10	784.61	135.81
ECG5000	94.39	94.27	93.93	94.75 ± 0.05	94.68 ± 0.09	93.55 ± 0.60	94.78 ± 0.06	3390.60	3390.60	2687.18	173.00
ECGFiveDays	99.48	99.77	100.00	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	2218.50	2218.50	205.96	124.82
ElectricDev	74.03	70.25	71.40	72.81 ± 0.25	72.72 ± 0.39	72.65 ± 0.38	72.63 ± 0.30	4363.10	4363.10	48 114.26	654.77
FaceAll	92.64	79.29	81.83	94.68 ± 0.40	94.64 ± 0.32	94.16 ± 0.67	94.14 ± 0.78	4660.60	4660.60	4373.65	196.47
FaceFour	98.86	95.45	95.45	97.61 ± 0.34	97.61 ± 0.34	98.41 ± 0.56	97.84 ± 0.34	1829.90	1829.90	321.47	125.82
FacesUCR	89.51	96.20	96.78	96.20 ± 0.09	96.20 ± 0.08	96.14 ± 0.16	96.34 ± 0.08	4842.70	4842.70	1599.40	183.20
FiftyWords	76.99	80.66	84.84	82.99 ± 0.41	82.99 ± 0.41	82.00 ± 0.47	82.46 ± 0.47	10 000.00	5000.00	8068.53	308.62
Fish	92.91	97.71	98.29	97.83 ± 0.62	98.00 ± 0.69	98.40 ± 0.56	98.74 ± 0.50	2257.90	1757.90	849.90	117.25
FordA	97.68	95.83	96.14	94.43 ± 0.28	94.04 ± 0.26	94.61 ± 0.18	94.05 ± 0.57	2812.60	2812.60	13 065.10	273.58
FordB	83.01	85.31	85.31	80.43 ± 0.78	79.81 ± 0.39	80.54 ± 0.36	80.52 ± 0.67	3086.00	3086.00	14 281.01	264.23
GunPoint	97.00	100.00	100.00	100.00 ± 0.00	100.00 ± 0.00	99.33 ± 0.00	100.00 ± 0.00	1829.80	1829.80	314.20	104.12
Ham	76.95	71.43	70.48	73.43 ± 1.16	71.62 ± 2.29	77.71 ± 1.29	73.52 ± 2.03	2292.30	2292.30	615.16	105.53
HandOutlines	91.57	95.41	95.41	94.11 ± 0.20	94.05 ± 0.30	94.00 ± 0.52	94.03 ± 0.46	2704.70	2704.70	3849.30	152.11
Haptics	51.56	56.49	56.82	52.11 ± 0.36	52.31 ± 0.75	52.50 ± 0.70	52.89 ± 0.91	3901.70	3901.70	742.73	131.02
Herring	60.47	76.56	71.88	69.53 ± 1.05	67.97 ± 1.60	65.16 ± 3.28	61.72 ± 1.88	2863.80	2863.80	407.39	104.51
InlineSkate	66.75	55.09	52.55	45.87 ± 0.65	45.49 ± 1.17	49.45 ± 0.88	48.42 ± 0.79	4943.00	2943.00	543.33	139.91
InsWinSou	66.80	63.33	64.44	65.66 ± 0.21	65.72 ± 0.24	66.12 ± 0.71	66.23 ± 0.43	2763.50	2763.50	1354.89	133.89
ItaPowDem	97.31	96.70	96.89	96.93 ± 0.09	96.82 ± 0.15	96.95 ± 0.19	96.88 ± 0.12	1050.70	1050.70	423.84	107.16
LarKitApp	80.64	89.07	89.60	90.00 ± 0.40	89.28 ± 0.88	89.01 ± 0.50	89.52 ± 0.68	3535.60	3535.60	1613.64	135.65
Lightning2	76.72	75.41	80.33	76.72 ± 0.66	76.72 ± 0.98	78.52 ± 2.59	80.33 ± 2.07	2626.50	2626.50	379.77	104.81
Lightning7	76.85	83.56	84.93	82.19 ± 0.61	82.88 ± 1.10	80.96 ± 1.88	82.47 ± 1.19	3695.90	3695.90	797.78	111.92
Mallat	96.57	95.86	96.42	95.63 ± 0.21	95.63 ± 0.21	92.99 ± 1.09	95.52 ± 0.26	10 000.00	5000.00	694.15	120.00
Meat	95.00	93.33	95.00	94.00 ± 2.00	93.83 ± 1.07	94.17 ± 0.83	94.50 ± 0.76	3599.30	3599.30	480.54	104.11
MedicalImages	81.67	78.55	80.13	79.67 ± 0.37	79.54 ± 0.40	79.05 ± 0.69	79.37 ± 0.67	5141.30	4141.30	2278.48	131.79
MidPhaOutAG	59.35	51.30	51.95	58.64 ± 0.71	61.56 ± 1.36	63.51 ± 2.23	56.17 ± 1.75	2795.10	2795.10	1707.16	114.49
MidPhaOutCor	83.61	84.88	83.85	83.47 ± 0.76	83.92 ± 1.01	83.51 ± 1.65	82.65 ± 1.13	3098.90	3098.90	2288.76	118.32
MiddlePTW	59.68	50.00	51.30	55.78 ± 0.74	54.35 ± 0.82	56.88 ± 1.79	54.87 ± 0.88	3797.90	3797.90	2012.32	119.74
MoteStrain	94.48	88.26	88.26	91.41 ± 0.30	91.53 ± 0.44	91.71 ± 0.29	91.38 ± 0.38	1835.30	1835.30	144.22	105.41
NonInvFECGT1[b]	93.64	95.98	96.08	95.25 ± 0.20	95.25 ± 0.20	95.80 ± 0.14	95.61 ± 0.20	10 000.00	5000.00	31 179.69	596.24
NonInvFECGT2[b]	94.60	96.08	96.69	96.77 ± 0.19	96.77 ± 0.19	96.74 ± 0.26	96.71 ± 0.18	10 000.00	5000.00	31 973.74	599.28
OliveOil[b]	90.00	80.00	76.67	91.33 ± 1.63	91.67 ± 1.67	93.00 ± 1.00	93.00 ± 1.00	3194.50	3194.50	299.23	243.63
OSULeaf[b]	84.88	95.45	95.04	94.01 ± 0.42	93.88 ± 0.90	94.17 ± 0.91	93.68 ± 0.92	2544.80	2544.80	1003.20	253.27
PhaOutCor[b]	84.06	84.03	84.27	82.96 ± 0.78	82.56 ± 0.74	82.10 ± 0.45	82.73 ± 0.79	3258.50	3258.50	7441.26	308.97
Phoneme[b]	39.76	33.70	33.81	28.05 ± 0.20	29.41 ± 0.37	24.69 ± 0.41	28.33 ± 0.42	4732.60	4732.60	2607.88	389.57
Plane[b]	100.00	100.00	100.00	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	7082.80	5582.80	577.74	251.66
ProPhaOutAG[b]	85.41	85.37	83.90	85.51 ± 0.22	85.66 ± 0.62	85.80 ± 0.67	85.27 ± 0.43	1705.40	1705.40	1752.40	243.38
ProPhaOutCor[b]	92.06	91.75	92.44	90.24 ± 0.60	89.14 ± 0.74	89.73 ± 0.42	90.65 ± 1.10	3019.40	3019.40	2312.24	248.79
ProPhaTW[b]	79.12	80.49	78.05	81.17 ± 0.66	81.07 ± 0.72	80.49 ± 0.79	79.07 ± 1.03	4931.30	3431.30	1998.84	254.29
RefDev[b]	59.04	49.60	52.00	53.49 ± 0.96	53.20 ± 0.87	54.08 ± 0.73	53.41 ± 1.23	3720.50	3720.50	1635.38	338.01
ScreenType[b]	54.61	58.93	57.60	48.56 ± 1.15	49.20 ± 2.09	46.99 ± 1.54	47.92 ± 0.97	3635.10	3635.10	1639.23	345.80
ShapeletSim[b]	97.89	75.00	100.00	100.00 ± 0.00	99.44 ± 0.86	100.00 ± 0.00	100.00 ± 0.00	2084.20	2084.20	159.30	229.72
ShapesAll[b]	86.12	91.33	92.83	90.72 ± 0.22	90.72 ± 0.22	90.18 ± 0.40	90.42 ± 0.30	10 000.00	5000.00	10 350.97	446.24
SmaKitApp[b]	82.35	77.07	77.60	81.60 ± 0.46	80.96 ± 1.03	82.27 ± 0.95	80.40 ± 0.96	3184.10	3184.10	1638.70	348.32
SonAIBORobS1[b]	89.58	83.53	84.86	92.26 ± 0.20	92.23 ± 0.39	94.26 ± 0.58	93.16 ± 0.32	1821.50	1821.50	156.69	236.41
SonAIBORobS2[b]	87.40	94.12	94.75	91.22 ± 0.27	91.22 ± 0.52	92.13 ± 0.36	92.24 ± 0.55	3025.80	3025.80	181.42	241.38
StarLightC[b]	97.94	97.67	97.56	98.06 ± 0.05	98.06 ± 0.04	98.01 ± 0.13	98.00 ± 0.13	4035.80	2535.80	4544.64	409.38
Strawberry[b]	96.84	98.38	98.38	98.14 ± 0.08	98.03 ± 0.34	97.62 ± 0.48	98.30 ± 0.27	3239.80	3239.80	2399.56	260.15
SwedishLeaf[b]	95.54	96.16	97.28	96.56 ± 0.29	96.32 ± 0.35	96.45 ± 0.34	96.50 ± 0.27	8106.60	7106.60	3604.54	300.89
Symbols[b]	97.24	98.29	97.99	97.43 ± 0.05	97.43 ± 0.05	97.84 ± 0.16	97.46 ± 0.06	10 000.00	5000.00	296.87	162.06
SynCon[b]	99.00	100.00	99.67	99.97 ± 0.10	99.73 ± 0.29	99.03 ± 0.10	99.83 ± 0.22	5810.70	5810.70	1495.16	264.37
ToeSeg1[b]	84.74	97.81	96.05	96.80 ± 0.34	96.62 ± 0.34	95.26 ± 1.25	95.88 ± 0.35	1071.80	1071.80	267.47	244.17
ToeSeg2[b]	87.92	93.85	93.08	92.08 ± 0.35	92.54 ± 1.24	93.77 ± 1.16	92.62 ± 0.92	1091.60	1091.60	258.58	243.47
Trace[b]	100.00	100.00	100.00	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	1826.30	1826.30	866.61	237.49
TwoLeadECG[b]	98.44	99.82	99.74	99.91 ± 0.00	99.91 ± 0.00	99.91 ± 0.00	99.91 ± 0.00	1840.70	1840.70	165.23	297.16
TwoPatterns[b]	99.69	100.00	100.00	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	1474.20	1474.20	4908.68	279.18
UWavGesLibA[b]	95.59	91.23	94.78	97.57 ± 0.08	97.36 ± 0.29	97.50 ± 0.17	97.56 ± 0.15	3914.80	3914.80	5160.15	362.54
UWavGesLibX[b]	82.88	82.94	83.08	85.50 ± 0.21	85.28 ± 0.47	85.14 ± 0.42	85.26 ± 0.28	6413.70	3413.70	5167.12	361.90
UWavGesLibY[b]	75.75	76.16	78.28	77.32 ± 0.23	76.99 ± 0.65	76.62 ± 0.51	77.26 ± 0.48	4142.30	2642.30	5208.30	362.11
UWavGesLibZ[b]	76.81	78.08	78.53	79.13 ± 0.20	78.67 ± 0.45	79.15 ± 0.27	78.96 ± 0.47	2373.20	1873.20	5176.96	361.32
Wafer[b]	99.97	99.90	99.92	99.83 ± 0.01	99.78 ± 0.06	99.86 ± 0.04	99.83 ± 0.06	1636.90	1636.90	4069.17	298.65
Wine[b]	77.78	66.67	64.81	80.93 ± 2.99	80.37 ± 3.72	83.15 ± 5.46	82.96 ± 4.96	2496.60	2496.60	364.49	240.98
WordSynonyms[b]	65.39	71.79	75.86	75.33 ± 0.27	75.34 ± 0.27	75.47 ± 0.73	76.00 ± 0.45	9825.40	5325.40	2422.28	317.08
Worms[b]	79.22	81.82	83.12	73.25 ± 1.04	73.25 ± 1.45	73.12 ± 1.65	72.21 ± 1.32	2563.70	2563.70	877.64	328.10
WormsTwoCla[b]	80.52	76.62	77.92	78.96 ± 1.82	79.35 ± 1.69	80.91 ± 1.65	77.79 ± 1.23	3403.80	3403.80	1026.81	324.07
Yoga[b]	85.59	91.77	92.80	91.16 ± 0.36	90.43 ± 0.43	91.38 ± 0.43	91.48 ± 0.52	2140.20	2140.20	1680.05	237.54

[a] The columns are taken from the corresponding Github repository, while the other results are averaged over 10 repeated experiments.

[b] The datasets are conducted on a Windows Server 2016 with two E5-2670 v3 CPUs, and the others are on a Ubuntu 18.04.6 Server with two Intel Xeon E5-2690 CPUs.

Table 10
Performance comparison on pruning ROCKET-PPV-MAX across 43 'Extra' UCR datasets. 'Acc.' stands for 'Accuracy'; '***AVERAGE***' represents the average over all listed datasets.

Dataset	r-STSF	L-Time	H-ITime	ROCKET & Pruning							
	[29]	[48]	[47]	Unpruned [11]	Pruned models' Acc. (%)			#Remaining kernels		Overall pruning time (s)	
	Acc.[a] (%)	Acc.[a] (%)	Acc.[a] (%)	ROCKET Acc.	S-ROCKET [13]	POCKET Stage 1	POCKET Stage 2	S-ROCKET [13]	POCKET	S-ROCKET [13]	POCKET
AVERAGE	–	84.94	85.39	85.18 ± 0.48	84.90 ± 0.72	85.29 ± 1.03	**85.48 ± 0.78**	4492.17	**3573.57**	3312.54	**271.23**
ACSF1[b]	90.40	91.00	94.00	87.90 ± 0.83	87.30 ± 1.35	89.80 ± 1.33	89.40 ± 1.74	6413.90	6413.90	614.10	257.82
AllGesWiiX[b]	–	76.57	79.14	77.34 ± 0.47	77.06 ± 1.08	75.47 ± 3.15	77.90 ± 0.95	3218.20	3218.20	1740.51	257.72
AllGesWiiY[b]	–	78.86	83.57	78.76 ± 0.60	76.21 ± 1.03	74.04 ± 1.38	77.80 ± 1.88	1353.20	1353.20	1740.39	259.80
AllGesWiiZ[b]	–	75.29	80.29	76.69 ± 0.50	76.00 ± 0.93	70.71 ± 3.41	75.49 ± 1.06	2993.50	1993.50	1729.77	263.38
BME[b]	99.87	99.33	99.33	100.00 ± 0.00	100.00 ± 0.00	99.93 ± 0.20	100.00 ± 0.00	1854.50	1854.50	251.77	233.15
Chinatown[b]	98.57	98.25	97.96	98.25 ± 0.00	98.25 ± 0.18	98.25 ± 0.00	98.51 ± 0.09	2266.10	2266.10	157.63	225.23
Crop[b]	77.70	75.73	77.21	76.64 ± 0.07	75.85 ± 1.27	73.21 ± 1.92	75.26 ± 0.74	1883.40	1383.40	66 160.18	826.16
DodgerLoopDay	–	57.50	58.75	60.50 ± 1.70	60.25 ± 1.84	60.88 ± 1.68	60.62 ± 2.11	5520.20	5020.20	512.74	154.22
DodgerLoopGame	–	81.88	83.33	86.09 ± 0.54	86.45 ± 0.80	88.33 ± 0.68	88.26 ± 0.63	1842.60	1842.60	199.85	123.50
DodLooWee	–	97.10	97.10	97.68 ± 0.29	97.54 ± 0.35	98.55 ± 0.00	98.48 ± 0.22	1827.50	1827.50	199.72	121.82
EOGHorSig	57.15	63.54	62.15	58.26 ± 1.05	58.01 ± 1.19	59.17 ± 1.46	58.45 ± 1.18	6933.10	6433.10	2627.53	168.83
EOGVerSig	52.82	41.16	46.69	54.70 ± 0.58	55.03 ± 0.91	55.28 ± 0.94	54.81 ± 0.58	5156.00	5156.00	2635.61	165.08
EthanolLevel[b]	61.84	74.00	78.80	58.56 ± 0.84	57.40 ± 0.80	58.86 ± 1.38	59.76 ± 0.92	2899.80	2899.80	2651.43	363.85
FreRegTra[b]	99.92	99.79	99.72	99.76 ± 0.03	99.69 ± 0.07	99.75 ± 0.04	99.77 ± 0.03	2959.60	2959.60	1094.73	250.27
FreSmaTra[b]	98.14	96.11	82.25	95.14 ± 0.18	94.92 ± 0.64	95.20 ± 3.17	96.24 ± 0.40	1833.70	1833.70	239.96	238.99
Fungi[b]	–	100.00	100.00	99.41 ± 0.16	99.41 ± 0.16	100.00 ± 0.00	99.78 ± 0.26	9756.90	5256.90	409.52	37.10
GestureMidAirD1[b]	–	75.38	75.38	77.85 ± 0.58	77.85 ± 0.58	74.92 ± 1.38	77.62 ± 0.80	10 000.00	5000.00	1898.31	345.22
GestureMidAirD2[b]	–	70.77	73.08	66.77 ± 1.02	66.77 ± 1.02	68.62 ± 1.37	67.15 ± 0.60	10 000.00	5000.00	1898.97	335.66
GestureMidAirD3[b]	–	42.31	42.31	39.62 ± 0.62	39.69 ± 0.62	39.69 ± 2.13	40.54 ± 0.98	9768.60	5268.60	1900.54	311.46
GesturePebbleZ1[b]	–	91.86	93.02	90.17 ± 0.31	90.29 ± 0.52	90.00 ± 0.35	90.23 ± 0.35	4825.90	4825.90	683.48	255.97
GesturePebbleZ2[b]	–	89.87	88.61	83.73 ± 0.85	83.61 ± 1.11	84.68 ± 1.05	84.56 ± 0.95	7076.70	7076.70	1185.65	263.11
GunPointAgeSpan[b]	98.99	99.05	99.68	99.02 ± 0.22	98.61 ± 0.60	97.12 ± 1.46	98.80 ± 0.37	1791.10	1791.10	1476.98	243.42
GunPoiMalVerFem[b]	100.00	99.05	100.00	99.78 ± 0.20	99.75 ± 0.19	100.00 ± 0.00	99.91 ± 0.15	963.80	963.80	1475.11	239.70
GunPoiOldVerYou[b]	100.00	97.78	97.78	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	1825.90	1825.90	1465.64	246.85
HouseTwenty[b]	91.93	98.32	97.48	96.30 ± 0.41	96.30 ± 0.41	96.30 ± 0.41	95.97 ± 0.50	2517.30	2517.30	523.17	244.51
InsEPGRegTra[b]	100.00	99.60	100.00	100.00 ± 0.00	99.96 ± 0.12	100.00 ± 0.00	100.00 ± 0.00	2114.60	2114.60	972.76	246.39
InsEPGSmaTra[b]	100.00	96.39	91.57	100.00 ± 0.00	99.88 ± 0.36	100.00 ± 0.00	100.00 ± 0.00	2114.40	2114.40	326.46	246.87
MelPed[b]	–	90.57	91.18	94.46 ± 0.23	94.16 ± 0.35	94.09 ± 0.41	94.12 ± 0.33	4981.40	4981.40	9637.49	311.78
MixShaRegTra[b]	94.90	98.06	97.65	97.06 ± 0.09	97.01 ± 0.11	97.13 ± 0.13	97.02 ± 0.11	3449.90	3449.90	3474.85	405.09
MixShaSmaTra[b]	89.59	94.39	92.16	93.78 ± 0.09	93.55 ± 0.26	93.88 ± 0.19	93.58 ± 0.12	2829.80	2829.80	1821.62	278.92
PicGesWiiZ[b]	–	78.00	78.00	79.40 ± 0.92	79.40 ± 0.92	82.20 ± 1.08	81.80 ± 1.66	10 000.00	5000.00	1404.72	284.39
PigAirPre[b]	37.50	47.12	56.73	80.05 ± 0.72	80.05 ± 0.72	84.71 ± 1.23	84.86 ± 0.62	10 000.00	5000.00	2412.84	248.78
PigArtPressure[b]	92.40	99.52	99.52	95.77 ± 0.47	95.77 ± 0.47	93.17 ± 0.52	94.81 ± 0.77	8835.50	5335.50	2374.94	232.38
PigCVP[b]	68.32	94.71	96.63	93.12 ± 0.22	93.12 ± 0.22	92.98 ± 0.24	93.03 ± 0.32	9764.00	5264.00	2383.67	230.76
PLAID[b]	–	91.99	94.23	81.38 ± 1.51	80.24 ± 2.05	84.00 ± 2.25	83.13 ± 1.67	3675.00	3675.00	4881.03	373.56
PowerCons[b]	100.00	96.11	93.89	98.56 ± 0.44	98.56 ± 1.03	99.83 ± 0.50	99.28 ± 0.70	784.10	784.10	1921.19	250.39
Rock[b]	75.80	86.00	84.00	68.20 ± 0.60	69.00 ± 1.84	80.00 ± 1.79	73.00 ± 3.00	4032.00	4032.00	411.11	256.21
SemHanGenCh2[b]	96.60	86.83	82.33	89.45 ± 0.62	88.40 ± 0.59	88.52 ± 0.80	88.47 ± 0.60	3053.50	3053.50	3310.66	352.21
SemHanMovCh2[b]	83.49	51.11	53.11	59.64 ± 1.06	57.76 ± 1.42	56.64 ± 3.23	55.09 ± 3.02	2924.80	2924.80	3785.82	369.80
SemHanSubCh2[b]	88.93	80.22	84.44	84.67 ± 0.82	83.33 ± 0.92	82.51 ± 1.54	83.27 ± 1.39	4115.40	4115.40	3307.84	358.55
ShaGesWiiZ[b]	–	96.00	92.00	92.20 ± 0.60	92.00 ± 0.89	93.80 ± 1.08	92.40 ± 1.20	7122.70	7122.70	1475.76	278.25
SmoothSubspace[b]	98.00	98.67	97.33	97.47 ± 0.27	97.27 ± 0.55	96.73 ± 0.36	97.67 ± 0.33	3084.50	3084.50	2391.30	254.11
UMD[b]	99.79	96.53	99.31	98.61 ± 0.00	98.82 ± 0.32	98.61 ± 0.00	98.61 ± 0.00	2800.30	2800.30	671.92	251.52

[a] The columns are taken from the corresponding Github repository, while the other results are averaged over 10 repeated experiments.

[b] The datasets are conducted on a Windows Server 2016 with two E5-2670 v3 CPUs, and the others are on a Ubuntu 18.04.6 Server with two Intel Xeon E5-2690 CPUs.

Appendix B. 43 'Extra' UCR datasets

See Table 10.

References

[1] Y. Huang, G.G. Yen, V.S. Tseng, Snippet policy network V2: Knee-guided neuroevolution for multi-lead ECG early classification, IEEE Trans. Neural Netw. Learn. Syst. (2022) 1–15.

[2] A. Bier, A. Jastrzebska, P. Olszewski, Variable-length multivariate time series classification using ROCKET: A case study of incident detection, IEEE Access 10 (2022) 95701–95715.

[3] S. Li, T. Li, C. Sun, X. Chen, R. Yan, WPConvNet: An interpretable wavelet packet kernel-constrained convolutional network for noise-robust fault diagnosis, IEEE Trans. Neural Netw. Learn. Syst. (2023) 1–15.

[4] H. Ismail Fawaz, B. Lucas, G. Forestier, C. Pelletier, D.F. Schmidt, J. Weber, G.I. Webb, L. Idoumghar, P.-A. Muller, F. Petitjean, Inceptiontime: Finding alexnet for time series classification, Data Min. Knowl. Discov. 34 (6) (2020) 1936–1962.

[5] S. Mauceri, J. Sweeney, M. Nicolau, J. McDermott, Dissimilarity-preserving representation learning for one-class time series classification, IEEE Trans. Neural Netw. Learn. Syst. (2023) 1–12.

[6] C. Sun, H. Li, M. Song, S. Hong, A ranking-based cross-entropy loss for early classification of time series, IEEE Trans. Neural Netw. Learn. Syst. (2023).

[7] H. Salehinejad, S. Sankar, J. Barfett, E. Colak, S. Valaee, Recent advances in recurrent neural networks, 2017, arXiv preprint arXiv:1801.01078.

[8] I. Revin, V.A. Potemkin, N.R. Balabanov, N.O. Nikitin, Automated machine learning approach for time series classification pipelines using evolutionary optimization, Knowl.-Based Syst. 268 (2023) 110483.

[9] B. Lucas, A. Shifaz, C. Pelletier, L. O'Neill, N. Zaidi, B. Goethals, F. Petitjean, G.I. Webb, Proximity forest: an effective and scalable distance-based classifier for time series, Data Min. Knowl. Discov. 33 (3) (2019) 607–635.

[10] M. Middlehurst, W. Vickers, A. Bagnall, Scalable dictionary classifiers for time series classification, in: Proceedings of the 20th Intelligent Data Engineering and Automated Learning, Part I 20, Springer, 2019, pp. 11–19.

[11] A. Dempster, F. Petitjean, G.I. Webb, ROCKET: exceptionally fast and accurate time series classification using random convolutional kernels, Data Min. Knowl. Discov. 34 (5) (2020) 1454–1495.

[12] A. Dempster, D.F. Schmidt, G.I. Webb, MINIROCKET: A very fast (almost) deterministic transform for time series classification, in: Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, 2021, pp. 248–257.

[13] H. Salehinejad, Y. Wang, Y. Yu, T. Jin, S. Valaee, S-ROCKET: Selective random convolution kernels for time series classification, 2022, arXiv preprint arXiv:2203.03445.

[14] H. Li, A. Kadav, I. Durdanovic, H. Samet, H.P. Graf, Pruning filters for efficient ConvNets, in: 5th International Conference on Learning Representations, 2017.

[15] S. Chen, W. Sun, L. Huang, WHC: Weighted hybrid criterion for filter pruning on convolutional neural networks, in: Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing, IEEE, 2023, pp. 1–5.

[16] L. Pantiskas, K. Verstoep, M. Hoogendoorn, H. Bal, Taking ROCKET on an efficiency mission: Multivariate time series classification with LightWaveS, in: 2022 18th International Conference on Distributed Computing in Sensor Systems, DCOSS, 2022, pp. 149–152.

[17] X. Liao, H. Li, L. Carin, Generalized alternating projection for weighted-2,1 minimization with applications to model-based compressive sensing, SIAM J. Imaging Sci. 7 (2) (2014) 797–823.

[18] 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. Discov. 31 (3) (2017) 606–660.

[19] A.P. Ruiz, M. Flynn, J. Large, M. Middlehurst, A. Bagnall, The great multivariate time series classification bake off: a review and experimental evaluation of recent algorithmic advances, Data Min. Knowl. Discov. 35 (2) (2021) 401–449.

[20] M. Shokoohi-Yekta, B. Hu, H. Jin, J. Wang, E. Keogh, Generalizing DTW to the multi-dimensional case requires an adaptive approach, Data Min. Knowl. Discov. 31 (2017) 1–31.
[21] Y.-S. Jeong, M.K. Jeong, O.A. Omitaomu, Weighted dynamic time warping for time series classification, Pattern Recognit. 44 (9) (2011) 2231–2240.
[22] G.E. Batista, E.J. Keogh, O.M. Tataw, V.M. De Souza, CID: an efficient complexity-invariant distance for time series, Data Min. Knowl. Discov. 28 (2014) 634–669.
[23] J. Lines, A. Bagnall, Time series classification with ensembles of elastic distance measures, Data Min. Knowl. Discov. 29 (2015) 565–592.
[24] Y. Wu, Q. Hu, Y. Li, L. Guo, X. Zhu, X. Wu, OPP-Miner: Order-preserving sequential pattern mining for time series, IEEE Trans. Cybern. 53 (5) (2023) 3288–3300.
[25] Y. Wu, X. Zhao, Y. Li, L. Guo, X. Zhu, P. Fournier-Viger, X. Wu, OPR-Miner: Order-preserving rule mining for time series, IEEE Trans. Knowl. Data Eng. 35 (11) (2023) 11722–11735.
[26] Y. Wu, Y. Meng, Y. Li, L. Guo, X. Zhu, P. Fournier-Viger, X. Wu, COPP-Miner: Top-k contrast order-preserving pattern mining for time series classification, IEEE Trans. Knowl. Data Eng. 36 (6) (2024) 2372–2387.
[27] H. Deng, G. Runger, E. Tuv, M. Vladimir, A time series forest for classification and feature extraction, Inform. Sci. 239 (2013) 142–153.
[28] J. Lines, S. Taylor, A. Bagnall, HIVE-COTE: The hierarchical vote collective of transformation-based ensembles for time series classification, in: Proceedings of the IEEE 16th International Conference on Data Mining, 2016, pp. 1041–1046.
[29] N. Cabello, E. Naghizade, J. Qi, L. Kulik, Fast, accurate and explainable time series classification through randomization, Data Min. Knowl. Discov. 38 (2) (2024) 748–811.
[30] L. Ye, E. Keogh, Time series shapelets: a novel technique that allows accurate, interpretable and fast classification, Data Min. Knowl. Discov. 22 (2011) 149–182.
[31] T. Rakthanmanon, E. Keogh, Fast shapelets: A scalable algorithm for discovering time series shapelets, in: Proceedings of the 2013 SIAM International Conference on Data Mining, SIAM, 2013, pp. 668–676.
[32] J. Hills, J. Lines, E. Baranauskas, J. Mapp, A. Bagnall, Classification of time series by shapelet transformation, Data Min. Knowl. Discov. 28 (2014) 851–881.
[33] J. Grabocka, N. Schilling, M. Wistuba, L. Schmidt-Thieme, Learning time-series shapelets, in: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2014, pp. 392–401.
[34] A. Dempster, D.F. Schmidt, G.I. Webb, HYDRA: Competing convolutional kernels for fast and accurate time series classification, Data Min. Knowl. Discov. (2023) 1–27.
[35] N. Tabassum, S. Menon, A. Jastrzebska, Time-series classification with SAFE: Simple and fast segmented word embedding-based neural time series classifier, Inf. Process. Manage. 59 (5) (2022) 103044.
[36] P. Schäfer, The BOSS is concerned with time series classification in the presence of noise, Data Min. Knowl. Discov. 29 (2015) 1505–1530.
[37] M. Middlehurst, J. Large, G. Cawley, A. Bagnall, The temporal dictionary ensemble (TDE) classifier for time series classification, in: Proceedings of Machine Learning and Knowledge Discovery in Databases: European Conference, Part I, Springer, 2021, pp. 660–676.
[38] J. Lines, S. Taylor, A. Bagnall, Time series classification with HIVE-COTE: The hierarchical vote collective of transformation-based ensembles, ACM Trans. Knowl. Discov. Data 12 (5) (2018) 1–35.
[39] M. Middlehurst, J. Large, A. Bagnall, The canonical interval forest (CIF) classifier for time series classification, in: Proceedings of 2020 IEEE International Conference on Big Data, IEEE, 2020, pp. 188–195.
[40] M. Middlehurst, J. Large, M. Flynn, J. Lines, A. Bostrom, A. Bagnall, HIVE-COTE 2.0: a new meta ensemble for time series classification, Mach. Learn. 110 (11–12) (2021) 3211–3243.
[41] R. Zuo, G. Li, B. Choi, S.S. Bhowmick, D.N.-y. Mah, G.L. Wong, SVP-T: A shape-level variable-position transformer for multivariate time series classification, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 37, 2023, pp. 11497–11505.
[42] Z. Xiao, X. Xu, H. Zhang, E. Szczerbicki, A new multi-process collaborative architecture for time series classification, Knowl.-Based Syst. 220 (2021) 106934.
[43] N.M. Foumani, C.W. Tan, G.I. Webb, M. Salehi, Improving position encoding of transformers for multivariate time series classification, Data Min. Knowl. Discov. 38 (1) (2024) 22–48.
[44] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, P.-A. Muller, Deep learning for time series classification: a review, Data Min. Knowl. Discov. 33 (4) (2019) 917–963.
[45] Q. Wen, T. Zhou, C. Zhang, W. Chen, Z. Ma, J. Yan, L. Sun, Transformers in time series: A survey, in: International Joint Conference on Artificial Intelligence, 2023.
[46] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna, Rethinking the inception architecture for computer vision, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 2818–2826.
[47] A. Ismail-Fawaz, M. Devanne, J. Weber, G. Forestier, Deep learning for time series classification using new hand-crafted convolution filters, in: 2022 IEEE International Conference on Big Data, Big Data, 2022, pp. 972–981.
[48] A. Ismail-Fawaz, M. Devanne, S. Berretti, J. Weber, G. Forestier, LITE: Light inception with boosting techniques for time series classification, in: 2023 IEEE 10th International Conference on Data Science and Advanced Analytics, DSAA, 2023, pp. 1–10.
[49] C.W. Tan, A. Dempster, C. Bergmeir, G.I. Webb, MultiROCKET: multiple pooling operators and transformations for fast and effective time series classification, Data Min. Knowl. Discov. 36 (5) (2022) 1623–1646.
[50] C.W. Tan, C. Bergmeir, F. Petitjean, G.I. Webb, Time series extrinsic regression: Predicting numeric values from time series data, Data Min. Knowl. Discov. 35 (2021) 1032–1060.
[51] K. Schlegel, P. Neubert, P. Protzel, HDC-MiniROCKET: Explicit time encoding in time series classification with hyperdimensional computing, in: 2022 International Joint Conference on Neural Networks, IEEE, 2022, pp. 1–8.
[52] B. Liu, Z. Han, X. Chen, W. Shao, H. Jia, Y. Wang, Y. Tang, A novel compact design of convolutional layers with spatial transformation towards lower-rank representation for image classification, Knowl.-Based Syst. 255 (2022) 109723.
[53] W. Sun, S. Chen, L. Huang, H.C. So, M. Xie, Deep convolutional neural network compression via coupled tensor decomposition, IEEE J. Sel. Top. Sign. Proces. 15 (3) (2021) 603–616.
[54] S. Chen, J. Zhou, W. Sun, L. Huang, Joint matrix decomposition for deep convolutional neural networks compression, Neurocomputing 516 (2023) 11–26.
[55] Y. Liu, M.K. Ng, Deep neural network compression by tucker decomposition with nonlinear response, Knowl.-Based Syst. 241 (2022) 108171.
[56] B. Lyu, S. Wen, Y. Yang, X. Chang, J. Sun, Y. Chen, T. Huang, Designing efficient bit-level sparsity-tolerant memristive networks, IEEE Trans. Neural Netw. Learn. Syst. (2023).
[57] J. Huang, B. Xue, Y. Sun, M. Zhang, G.G. Yen, Split-level evolutionary neural architecture search with elite weight inheritance, IEEE Trans. Neural Netw. Learn. Syst. (2023).
[58] G. Lee, K. Lee, DNN compression by ADMM-based joint pruning, Knowl.-Based Syst. 239 (2022) 107988.
[59] J. Chang, Y. Lu, P. Xue, Y. Xu, Z. Wei, Iterative clustering pruning for convolutional neural networks, Knowl.-Based Syst. 265 (2023) 110386.
[60] I. Preet, O. Boydell, D. John, Class-separation preserving pruning for deep neural networks, IEEE Trans. Artif. Intell. (2022) 1–11.
[61] M.A. Omidi, B. Seyfe, S. Valaee, Reducing the computational complexity of learning with random convolutional features, in: Proceedings of 2023 IEEE International Conference on Acoustics, Speech and Signal Processing, 2023, pp. 1–5.
[62] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, et al., Distributed optimization and statistical learning via the alternating direction method of multipliers, Found. Trends Mach. Learn. 3 (1) (2011) 1–122.
[63] W. Deng, W. Yin, Y. Zhang, Group sparse optimization by alternating direction method, in: Wavelets and Sparsity XV, Vol. 8858, SPIE, 2013, pp. 242–256.
[64] J. Sherman, W.J. Morrison, Adjustment of an inverse matrix corresponding to a change in one element of a given matrix, Ann. Math. Stat. 21 (1) (1950) 124–127.
[65] Y. He, G. Kang, X. Dong, Y. Fu, Y. Yang, Soft filter pruning for accelerating deep convolutional neural networks, in: Proceedings of the 27th International Joint Conference on Artificial Intelligence, 2018, pp. 2234–2240.
[66] H.A. Dau, E. Keogh, K. Kamgar, C.-C.M. Yeh, Y. Zhu, S. Gharghabi, C.A. Ratanamahatana, Yanping, B. Hu, N. Begum, A. Bagnall, A. Mueen, G. Batista, M.L. Hexagon, The UCR time series classification archive, 2018.
[67] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in python, J. Mach. Learn. Res. 12 (2011) 2825–2830.