Appendix

Adacket: ADAptive Convolutional KErnel Transform for Multivariate Time Series Classification

Junru Zhang, Lang Feng, Haowen Zhang, Yuhan Wu, Yabo Dong

A Details of Method

A.1 Pseudocodes of State Space

Algorithm A.1: Encode the state space $\mathcal{S}$ for RL agent

Input: Time series $\mathbf{X}$, current time step s, previous state $\mathbf{s}_{s-1}$, the total number of kernel parameters $\mathbf{p}^{conv}$, the total number of output channels $\mathbf{p}^{out}$

Output: Current state $\mathbf{s}_s$

$z_s \leftarrow$ empty vector
$c(s) \leftarrow (s-1) \bmod F + 1$
Add $c(s)$ to $\mathbf{z}_s$
for $k \in \{7, 9\}$ **do**
  for $i \in N$ **do**
    $\mathbf{u} \leftarrow \text{Conv1D}(\mathbf{x}_{i,c(s)}, \text{kernel\_size} = k)$
    $\text{Max} \leftarrow \max(\mathbf{u})$
    $\text{PPV} \leftarrow \frac{\sum_{i=0}^{T-1}[u_i > 0]}{T}$
    $\mathcal{A}(\mathbf{u})$ with frequency units $\{f_1, f_2, ..., f_Q\} \leftarrow \text{FFT}(\mathbf{u})$
    $\text{Centroid} \leftarrow \frac{\sum_{i=0}^{Q-1} f_i |\mathcal{A}(\mathbf{u})_i|}{\sum_{i=0}^{Q-1} |\mathcal{A}(\mathbf{u})_i|}$
    $\text{PeakFactor} \leftarrow (\max(|\mathcal{A}(\mathbf{u})_i|)/\sqrt{\frac{1}{Q}\sum_{i=0}^{Q-1} |\mathcal{A}(\mathbf{u})_i|^2})$
  end
  Average these four features along the instance direction
  Add four feature values to $\mathbf{z}_s$
end
if $s = 1$ **then**
  Add six zeros elements to $\mathbf{z}_s$
  $\mathbf{s}_0 \leftarrow$ zero matrix$(\alpha, 15)$;
end
else
  $g_{s-1} \leftarrow (p_{s-1}^{conv} - p_{s-2}^{conv})/(p_{s-1}^{conv} + 1e-6)$
  $g'_{s-1} \leftarrow (p_{s-1}^{out} - p_{s-2}^{out})/(p_{s-1}^{out} + 1e-6)$
  Receive the previous action $\mathbf{a}_{s-1}$
  Add g_{s-1}, g'_{s-1}, $a_{s-1,1}$, $a_{s-1,2}$, $a_{s-1,3}$ and $a_{s-1,4}$ to $\mathbf{z}_s$
end
Scale $\mathbf{z}_s$ to [0,1]
$\mathbf{s}_{s,1:\alpha-1} \leftarrow \mathbf{s}_{s-1,2:\alpha}$
$\mathbf{s}_{s,\alpha} \leftarrow \mathbf{z}_s$;

A.2 Pseudocodes of Action Space

Algorithm A.2: Determine a kernel-channel pair from the action space $\mathcal{A}$

Input: Time series $\mathbf{X}$, current time step s, hyperparameters β and κ
Output: A kernel-channel pair $(\mathbf{v}_s, \mathbf{c}_s)$

$c(s) \leftarrow (s-1) \bmod F + 1$
$a_{s,1}, a_{s,2}, a_{s,3}, a_{s,4} \leftarrow$ the policy of the agent
$h_s^{in} \leftarrow \lfloor \frac{F - c(s)}{(F - c(s)) \times a_{s,1}} \rfloor$
$\Delta \leftarrow \lfloor (F - c(s)) \times a_{s,1} \rfloor$
for $m \leftarrow 1$ **to** h_s^{in} **do**
  $c_{s,m} \leftarrow c(s) + (m-1) \times \Delta$
end
$h_s^{out} \leftarrow \lfloor \beta * a_{s,2} \rfloor$
$h_s^{ks} \leftarrow \lceil (a_{s,3} \times \kappa) \rceil$
$F_s \leftarrow \lceil a_{s,4} * T \rceil$
$h_s^{dil} \leftarrow \lceil \frac{F_s - h_s^{ks}}{h_s^{ks} - 1 + 1e-6} \rceil$
if $h_s^{out} = 0$ **then**
  $\mathbf{v}_s \leftarrow$ empty set;
end
else
  $\mathbf{v}_s \leftarrow$ Initialize kernels with a four-tuple $(h_s^{in}, h_s^{out}, h_s^{ks}, h_s^{dil})$
end
return $(\mathbf{v}_s, \mathbf{c}_s)$

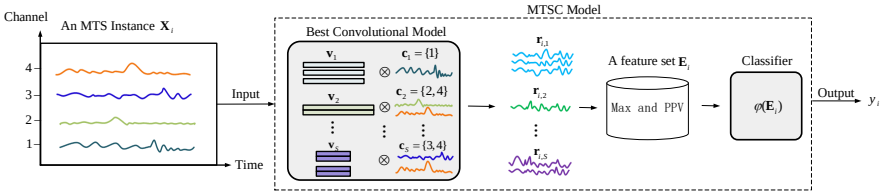

Fig. A.1. MTSC models designed by Adacket for feature extraction and classification. $\otimes$ represents a convolutional operation. The convolutional kernels from the same group are depicted using same-colored rectangles. The best convolutional model output by the agent provides the representations of the input instance $\mathbf{X}_i$ with four channels, and then the feature set $\mathbf{E}_i$ is constructed by extracting two features from each representation sequence $\mathbf{r}_{i,s}$. Finally, the feature set is fed to a linear classifier to obtain a precision result y_i. By replacing the random convolutional model of ROCKET with the convolutional model generated by Adacket, we create a new MTSC model.

A.3 The Entire Convolutional Model

As Fig. A.1 shows, the best convolutional model output by Adacket is combined with a feature extractor and a linear classifier, thus establishing a new convolutional model to complete the classification task.

B Experimental Details

B.1 Details in DDPG

Regarding the actor-critic network architecture, we set the learning rate of the actor network μ_θ to 1e-3 and the critic network Q_ξ to 1e-2. Following the standard settings in DDPG, we train the network with a batch size of 32 and a replay buffer size of 2000. To enhance the robustness and training efficiency of the actor-critic network, we set α to 5, which incorporates the past five historical observations with the current observation. To facilitate resource efficiency, We restrict the number of convolutional kernel groups to a maximum of 100, i.e., the total number of steps S set to 100. Adacket initially explores 5 episodes by randomly sampling actions between 0 and 1, and then exploits 30 episodes with a noise set $\mathcal{N}_s$, where each noise is drawn from a Gaussian distribution $\mathcal{N}(-0.1, 0.1)$.

B.2 Sensitivity Analysis

We explore the effect of ϵ, β and κ on model performance. In each case, only the given parameter (e.g., ϵ) is varied, keeping all other parameters fixed at their default values. The comparison is made on UEA 30 archive.

Trade-off Factor ϵ Adacket provides a solution to the trade-off between control performance and resource efficiency by adjusting the value of ϵ in the reward equation (Eq. 5 in paper). The average ranks of Adacket across various indicators, including accuracy, number of parameters, and memory usage, for different values of ϵ (i.e., 0.96, 0.97, 0.98, 0.99, and 1.00) are presented in Fig. B.2. The results indicate that in most cases, there are no significant differences in accuracy, parameter consumption, and memory usage among the models, suggesting that agents can reduce resource consumption without compromising accuracy. Specifically, when ϵ is set to 1.00, the agent prioritizes searching for models with better performance, leading to a focus on more precise convolutional kernels, and Adacket(1.00) achieves the best accuracy ranking in Fig. B.2 (a). Conversely, setting ϵ to 0.96 encourages the agent to explore a wider range of high-performance models with lower resource consumption, leading to Adacket(0.96) with the best ranking in resource efficiency as depicted in Fig. B.2 (b) and (c) It is noteworthy that when ϵ is set to 0.99, Adacket(0.99) achieves an accuracy ranking second only to Adacket(1.00), with no significant difference in accuracy observed between the two in Fig. B.2 (a). Furthermore, Adacket(0.99) achieves significantly better rankings in resource efficiency than Adacket(1.00), as shown in Fig. B.2

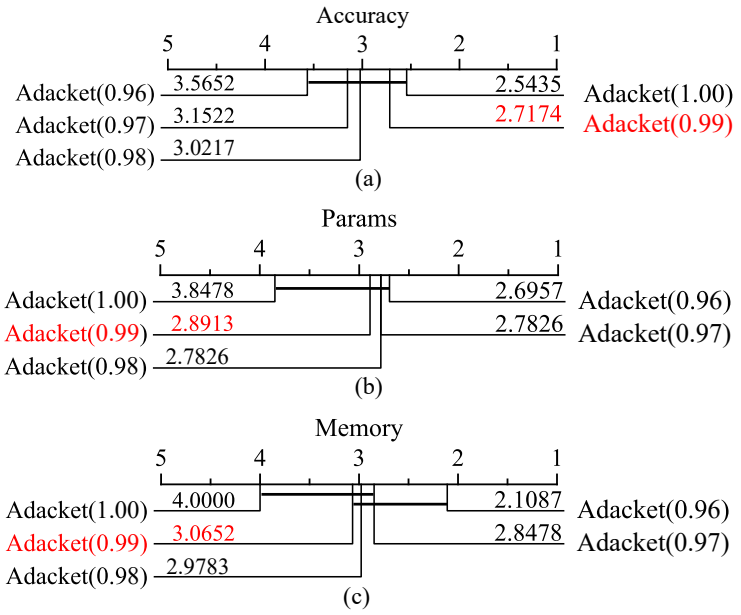

Fig. B.2. CD diagrams for comparing Adacket with different ϵ on UEA 30 datasets. The red mark indicates the default setting in the original paper.

(b) and (c). Hence, we adopt $\epsilon = 0.99$ to strike a reasonable balance between performance and resource consumption, thereby establishing resource-efficient conventional models. In conclusion, Adacket offers a flexible approach to managing the trade-off between performance and resources by adjusting ϵ values. Our results demonstrate that by appropriately setting ϵ, agents can effectively reduce resource consumption while maintaining comparable accuracy, leading to more resource-efficient and practical models.

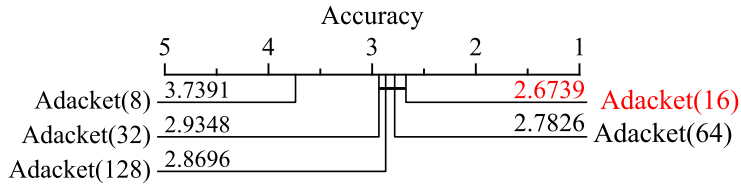

Fig. B.3. CD diagrams for comparing Adacket with different β on UEA 30 datasets. The red mark indicates the default setting in the original paper.

Upper bound of the output channel β We evaluate increasing the number of output channels for each group of convolutional kernels between 8 and 128. Fig. B.3

illustrates the impact of β on accuracy. The results reveal that, with the exception of Adacket(8), there is no significant difference in accuracy across the other models. This suggests that, in most cases, the agent can identify an appropriate number of output channels within the predetermined search space. However, if the value of β is too small, it causes the search space to be extremely limited, thus impeding the agent's ability to explore a broader range of output channels. Moreover, when $\beta = 16$, Adacket(16) achieves the highest rank, indicating that the agent can more efficiently optimize the number of output channels within this particular search space.

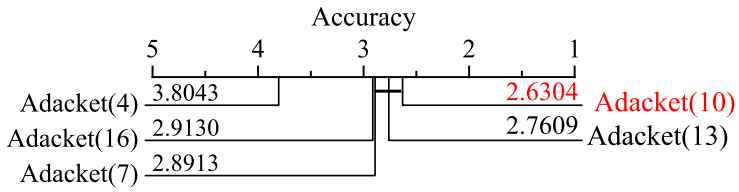

Fig. B.4. CD diagrams for comparing Adacket with different κ on UEA 30 datasets. The red mark indicates the default setting in the original paper.

Upper bound of the kernel size κ We vary the maximum kernel size κ (i.e., 4,7, 10, 13, and 16) in Adacket to compare accuracy across models. As shown in Fig. B.4, we observe no significant differences in accuracy among the models except for Adacket(4), which verified that agents can find suitable kernel size within the search space in most cases. Setting κ too low limits the search range and hinders the agent's ability to find the optimal kernel size. Furthermore, Adacket(10) achieves the highest ranking when $\kappa = 10$, indicating that this setting can encourage the agent to find out valuable kernel sizes more efficiently. Additionally, this setting can ensure that the kernel sizes discovered by the agent are typically smaller than the dataset length. Therefore, we set κ to 10.

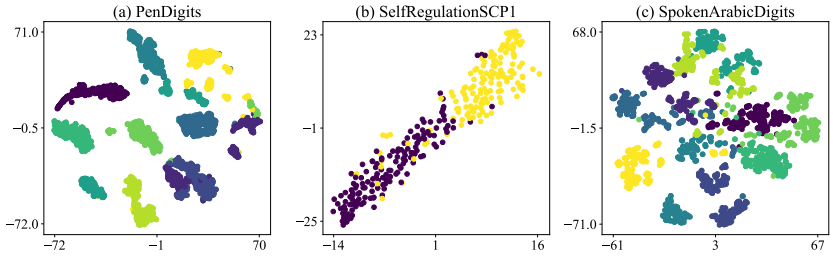

Fig. B.5. Two-dimensional T-SNE with perplexity 30 of the learned representations on the three UEA datasets. Different colors represent different classes.

B.3 Visualization of Representations

To assess the clusterability of the learned representations, we visualize the feature distribution adopting T-SNE [1]. A well-learned and encoded latent state information should result in representations that are closely clustered together in the encoding space, corresponding to the same underlying state. Fig. B.5 displays the distribution of the representations learned by our model, demonstrating the ability to distinguish between different classes in the latent space.

C Discussion and Future Work

Adacket employs adaptive convolution kernel transformations to tackle challenges in multivariate time series classification (MTSC) and showcases promising results on the public UEA MTSC archive. Utilizing a reinforcement learning framework, Adacket autonomously searches for efficient convolutional kernels by maximizing multi-objective rewards. In contrast to constrained optimization approaches, such as gradient descent-based methods, Adacket extensively explores the design space, fundamentally constructing lightweight models that enhance the performance of MTSC tasks. However, gradient descent-based methods do not directly optimize network structures, leaving them unable to significantly reduce computational load and memory costs.

Diverging from standard convolutional neural networks, which perform uniform operations across all channels, Adacket adopts agent-guided actions to adaptively design convolutional kernels in both channel and temporal dimensions. This strategy is rooted in the heterogeneous nature of diverse time series datasets, where valuable channels and their interactions vary across datasets, as validated by the experimental evidence in Fig. 3 of our paper. Furthermore, in the context of individual datasets, optimal combinations of input channels and their corresponding time scales exhibit disparities, as depicted in Fig. 4. These findings reinforce Adacket's approach to designing convolution kernels that cater to both channel and temporal dimensions. The results outperforming standard convolutional neural networks confirm the efficacy of our processing method. To address the computational bottleneck of reinforcement learning (as discussed in Sec. 2.2 of the paper), we substitute classification accuracy with contrastive loss. Fig. B.5 concurrently illustrates the agent's ability to effectively utilize selected convolutional kernels to produce meaningful representations.

Moreover, Adacket offers numerous fascinating applications and research topics. For example, besides using the maximum step S as the termination condition for each episode, alternative termination conditions, such as computational budget constraints (e.g., training time, resource efficiency), could be considered for substitution or addition. By integrating the hyperparameter ϵ, the balance between performance and efficiency can be flexibly adjusted within end-users defined budget constraints, providing intriguing directions for future research. Meanwhile, we are dedicated to performing comprehensive comparisons and analyses with an expanded range of MTSC methods. Importantly, we hope our work will inspire further studies into developing significant convolutional models

for time series analysis, ultimately addressing challenges arising from diverse and complex real-world applications.

References

1. Van der Maaten, L., Hinton, G.: Visualizing data using t-sne. Journal of machine learning research **9**(11) (2008)