S-Rocket: Selective Random Convolution Kernels for Time Series Classification

Hojjat Salehinejad, Member, IEEE, Yang Wang, Yuanhao Yu, Tang Jin, and Shahrokh Valaee, Fellow, IEEE

Abstract—Random convolution kernel transform (Rocket) is a fast, efficient, and novel approach for time series feature extraction using a large number of independently randomized initialized 1-D convolution kernels of different configurations. The output of the convolution operation on each time series is represented by a partial positive value (PPV). A concatenation of PPVs from all kernels is the input feature vector to a Ridge regression classifier. Unlike typical deep learning models, the kernels are not trained and there is no weighted/trainable connection between kernels or concatenated features and the classifier. Since these kernels are generated randomly, a portion of these kernels may not positively contribute in performance of the model. Hence, selection of the most important kernels and pruning the redundant and less important ones is necessary to reduce computational complexity and accelerate inference of Rocket for applications on the edge devices. Selection of these kernels is a combinatorial optimization problem. In this paper, we propose a scheme for selecting these kernels while maintaining the classification performance. First, the original model is pre-trained at full capacity. Then, a population of binary candidate state vectors is initialized where each element of a vector represents the active/inactive status of a kernel. A population-based optimization algorithm evolves the population in order to find a best state vector which minimizes the number of active kernels while maximizing the accuracy of the classifier. This activation function is a linear combination of the total number of active kernels and the classification accuracy of the pre-trained classifier with the active kernels. Finally, the selected kernels in the best state vector are utilized to train the Ridge regression classifier with the selected kernels. This approach is evaluated on the standard time series datasets and the results show that on average it can achieve a similar performance to the original models by pruning more than 60% of kernels. In some cases, it can achieve a similar performance using only 1% of the kernels.

Index Terms—Convolution kernels, feature selection, population-based optimization, pruning, time series classification.

I. INTRODUCTION

A regular time series is generally defined as a sequence of recorded observations through time/index in which the spacing of observation times/indices is constant. In an irregular time series, the spacing between observations is not constant. Mostly, time series refers to a regular time series in the literature [1]. Time series occurs in various real-world applications such as human activity recognition [2], travel mode detection [3], natural language processing [4], speech recognition [5], shopping pattern recognition [6], electronic health records [7], and medical imaging [8].

Acknowledging availability of certain massive time series datasets, most recent state-of-the-art methods for time series classification are based on learning a large number of parameters such as in recurrent neural networks [9]. Generally, these methods are computationally expensive, challenging to scale, and require significant amount of data and training time [10]. Many real-world applications, such as oscillogram signal classification [11] and human activity recognition [2] applications have access to limited-imbalanced time series data, which further complicates training of learning models.

Challenges in training very large learning models for time series have motivated the development of more scalable and much faster time series classification models such as random convolution kernel transform (Rocket) [10], MiniRocket [12], InceptionTime [13] and Proximity Forest [14]. Among these and many other methods, Rocket and MiniRocket have shown significant performance in classification of time series with noticeably less computational time.

Rocket [10] and MiniRocket [12] use random convolution kernels for transforming input time series into a set of features to train a linear classifier, without training the kernels. Both methods have shown fast and accurate time series classification on standard datasets and for different applications in a fraction of the training time of existing methods, such as UCR archive [15], inter-burst detection in electroencephalogram (EEG) signals [16], driver’s distraction detection using EEG signals [17], functional near infrared spectroscopy signals classification [18], and human activity recognition [19].

Similar to many machine learning models, Rocket uses a very large number of parameters (not trained), which can be prone to overparameterization if the number of parameters in the model exceeds the size of the training samples [20]. It is unclear which kernels are more effective in achieving a high classification performance. By identifying these kernels and pruning the redundant and inefficient ones, it is possible to reduce computational complexity of the model for faster inference, particularly on devices with limited resources such as edge devices.

In general, there are three approaches in solving the overparameterization problem which are structural efficiency, quantization, and pruning [21]. Searching for the best subset of convolution kernels (and hence the corresponding features) is
an NP-hard combinatorial optimization problem [21], [22]. Markov chain Monte Carlo (MCMC) [23] and simulated annealing (SA) are two popular methods for solving NP-hard combinatorial problems. We have proposed a binary differential evolution (BDE) [24] for pruning deep neural networks [24], [25], which can search the optimization landscape in parallel and handle multiple optimization constraints.

Most of the pruning and neural architecture search (NAS) methods are focused on pruning trainable convolution kernels. Unlike typical deep neural networks, the random convolution kernel transform approach in Rocket uses 1-D kernels to generate features from time series without training the kernels. Hence, this structure is different from a multi-layer (deep) feature extraction approach in deep learning. In this paper, we propose S-Rocket for reducing the computational complexity of Rocket by selecting the efficient convolution kernels [10]. To best of our knowledge, this is the first attempt in the literature to prune redundant and less efficient kernels for the Rocket models in classification of time series.

S-Rocket has three main steps which are pre-training, optimization, and post-training. First, Rocket is fully trained using the training data. Unlike most typical pruning methods which use a greedy search or statistical evaluation approach, S-Rocket uses the trained classifier at full capacity as the classification accuracy objective function. In the optimization step, similar to our proposed BDE approach in [26], a pool of candidate state vectors is used where the binary state of each element in a vector is a representation of an active/inactive kernel. A linear combination of the accuracy of the classifier (trained in step 1) for sparse input features and the number of active kernels per state vector is the objective function to minimize. Finally, the classifier is retrained from scratch with the masked input features by the best state vector found in the optimization step.

II. BACKGROUND

A. Overparameterization

There are three main non-mutually exclusive approaches to address the overparameterization problem in neural networks which can also be generalized for other machine learning models. These approaches are structural efficiency, quantization, and pruning. The structural efficiency approach can be divided into five categories which are knowledge distillation, special matrix structures, manually designed architectures, neural architecture search, and weight sharing [21]. Some of these approaches are described below.

1) Quantization: The feature values and weights in a model are generally stored as 32-bit floating-point. Storage and computation of these values require dedicated resources and can result in more energy consumption and slower training and inference, particularly for models with a very large number of floating-point operations on resource-limited devices. Quantization can reduce the number of bits used for the representation of the weights and the feature values [27].

2) Manually Designed Architectures: One of the most common approaches for increasing the efficiency of learning models is redesigning different building blocks and architecture of the model. For instance, the global average pooling reduces the spatial dimension of each channel into a single feature by averaging over all values within a channel [28]. Another example is SqueezeNet [29], a redesign of the AlexNet [30], which reduces the number of channels by implementing $1 \times 1$ convolutions.

3) Neural Architecture Search: This approach is broadly used to design models using optimization and search approaches. The optimization problems are generally designed in a discrete space of possible architectures (states) with an objective (or multi-objective) function [27]. Evaluation of the objective function for all possible states is an NP-hard problem [21].

4) Pruning: Pruning refers to permanently removing a subset of a model’s parameters. This approach can reduce overfitting (particularly in limited data [31]), tackle the overparameterization problem [32], [33], and increase resource efficiency of neural networks. Most pruning methods are applied after fully training a model, followed by retraining the pruned model [27]. From an architectural perspective, pruning methods are divided into unstructured and structured approaches. Unstructured pruning does not follow a specific geometry and happens at channel, filter, and intra-filter levels and the geometrically sparse weights are difficult to implement in practice. Structured pruning typically follows a geometric structure and removes a subset of weights such as the entire filter [34] where compared with the unstructured approach has very little computational cost overhead [21].

Pruning generally targets two objectives which are reducing the number of parameters and increasing the classification performance. Hence, a major challenge is selection of a subset of the model’s parameters without dropping the classification performance, which creates a dilemma between performance and model’s size. Generally, pruning can reduce the size of a model but may not improve the efficiency in terms of training or inference time [35].

Performance evaluation of neural networks in pruning approaches is generally performed using the Softmax function (which is a form of Gibbs distribution). As it is noted in the Knowledge Distillation work by Hinton et. al. [36], the temperature factor in the Gibbs distribution can provide a control on softness of the probability distribution over target classes. In addition, computing the partition function is a computing bottleneck, particularly for applications with very large number of target classes (a.k.a. extreme classification problems [37], [38]). The larger the number of classes, the lower the precision of probability values over the target classes, which may encourage learning noise and the gradients vanishing [37], [39].

Layer-wise pruning with manual setup of a sensitivity parameter per layer, is a common approach which dictates re-visiting and adjusting these parameters during fine-tuning [35]. One of the main reasons for a layer-wise pruning approach is the lower computational complexity, compared with a network-wise approach. The common trend to evaluate the
importance of parameters is to treat all layers in the network similarly, particularly for threshold-based methods. Pruning all layers uniformly tends to perform worse than intelligently allocating parameters to different layers \[^{[33]}\]. The \(l_1\) and \(l_2\) norms of the weights are typical methods to detect importance of parameters which requires more iterations to converge than general methods \[^{[35]}\]. Mutual information has been used as a metric to measure the strength of the relationship between filters of adjacent layers, across every pair of layers \[^{[40]}\].

B. Evolutionary Pruning

Pruning modern learning models is an NP-hard combinatorial optimization problem. As an example, AlexNet \[^{[30]}\] has 96 filters in the first convolutional layer which corresponds to 2\(^{96}\) pruning possibilities. Deterministic approaches are not able to find a guaranteed solution to this optimization problem. In addition, these methods cannot handle different constraints and pruning criteria in the objective function. Population-based global optimization methods \[^{[41]}\] can handle a wide variety of optimization constraints (even at multi-objective \[^{[42]}\] and many-objective \[^{[43]}\] scales) and can provide at least a feasible solution.

Genetic algorithm (GA) is one of the earliest methods for discovering a combination of connections for enhancing training of multi-layer perceptron models (MLPs) \[^{[44]}\]. GA, particularly with a small population size, has showed a better performance than the Bayesian models \[^{[45]}\]. In a multi-objective GA approach \[^{[46]}\], the objective function is to minimize a linear combination of the weighted average of network loss, computational complexity, and sparsity, where each parameter is controlled by a coefficient.

Differential evolution (DE) is another popular population-based method for pruning weights in deep learning models \[^{[26, 47]}\]. We have previously proposed energy-based objective functions for dropout and pruning of deep neural networks using BDE. Ising energy objective functions are proposed in \[^{[48]}\] and \[^{[49]}\], which represent the saturation and activation level of neurons in an MLP network. The \(ED\) method uses energy-based models (EBMs) \[^{[50]}\] as a measure of compatibility, which represents the dependencies of a subset of the network variables as a scalar energy, based on the definition of EBMs. In this scheme, a BDE algorithm evolves a set of binary state vectors, to minimize the energy-based objective function. DE is also used in \[^{[47]}\] to solve an optimization problem corresponding to sparsity and network accuracy.

C. Random Convolution Kernel Transform

It is shown that utilizing the combination of random convolutional filters with rectification, pooling, and local normalization for feature extraction can have a similar performance to learned features in small networks and datasets \[^{[51]}\]. This approach was also implemented in \[^{[52]}\] for different number of layers on larger networks and larger datasets, in a dissimilar setup to \[^{[51]}\]. It is shown that the classification performance drops in the first two layers and it gets to near random in layer three and subsequent ones \[^{[52]}\].

Shallow random networks, initialized as a bank of arbitrary randomized nonlinearities, called Convolutional Kitchen Sinks (CKS) \[^{[53]}\], has achieved good classification results, particularly for transcription factor binding site prediction for DNA sequences using only a one layer random convolutional neural network and a linear classifier \[^{[54]}\]. All the parameters of this setup are independent and identically distributed and randomly selected from a Gaussian distribution with a fixed variance.

The conventional training methods for convolution kernels use gradient-descent searching which is generally a time-consuming task prone to various challenges. In contrast to the learned convolution kernels in typical convolutional neural networks, these kernels are randomly initialized without learning in Rocket models. Rocket leverages different aspects of a convolution kernel with respect to the values of the weights (including the bias term), length, dilation, and padding. A bank of randomly generated kernels with respect to these factors, can get astonishing results, particularly with no/few labeled samples \[^{[51]}\].

Fig. 1(a) shows different steps of the Rocket model. It initializes a bank of random convolution kernels (e.g. 10,000 kernels is suggested in \[^{[10]}\]) where convolution of each kernel with an input time series produces a feature vector. Each feature vector is then represented by the proportion of positive values (PPV) and/or the maximum value \[^{[10]}\]. Later, the concatenation of PPV values from the kernels is used as the input feature vector to train a Ridge regression classifier. As an example, Figure 1(b) shows a sample time series from the UCR \[^{15}\] dataset and the corresponding PPV values are presented in Figure 1(c). This Figure shows that the length
of the input feature vector to the classifier is 10,000 which corresponds to the number of 1-D kernels. In fact, regardless of the length of the time series, it is represented by a feature vector with the length equal to the number of kernels. This approach eliminates padding of time series to identical lengths and generalizes the classifier for various-length time series.

One major parameter in initializing kernels is dilation, which works like a sampler that spreads a kernel over an input signal (e.g. in a dilation of three, every third element of the input signal is convolved with the kernel) \cite{55}. Rocket initializes its weights from a Normal distribution \( \mathcal{N}(0, 1) \) and the bias term from a uniform distribution \( \mathcal{U}(-1, 1) \). The design of MiniRocket follows a \textit{structural efficiency} approach in tackling the overparameterization problem in Rocket, such that it is manually modified for faster and more accurate performance.

III. S-ROCKET MODEL

Rocket \cite{10} and MiniRocket \cite{12} randomly initialize a large number of convolution kernels for feature extraction. We propose S-Rocket for selecting the most efficient kernels with respect to maintaining the classification performance of the original model while reducing the computational complexity of the original model. S-Rocket is discussed for Rocket and a similar procedure is also applicable to MiniRocket without loss of generality.

Let \( \{(x_1, y_1), \ldots, (x_N, y_N)\} \) represent the training data, where \( x_n \) is a time series and \( y_n \) is the corresponding label. S-Rocket has three main steps which are pre-training, optimization, and post-training, as demonstrated in Figure \ref{fig:diagram}

![Fig. 2: Diagram of the pre-training, optimization, and post-training steps in S-Rocket (without loss of generality for S-MiniRocket). \( \circ \) is the element-wise product.](image)

Algorithm \cite{1} also shows the pseudocode of the S-Rocket.

A. Pre-Training

1) Kernels Initialization: A set of \( D \) random convolution kernels \( \Theta = \{w_1, \ldots, w_D\} \) is initialized, as recommended in \cite{10}, such that for each kernel \( w \in \Theta \) the following values are set:

- \textit{Length}: The length of each kernel is randomly selected from \( \{7, 9, 11\} \) with equal probability.
- \textit{Weights}: The weights are randomly sampled from a Normal distribution \( \mathcal{N}(0, 1) \).
- \textit{Bias}: The bias value for each kernel is randomly chosen from a uniform distribution \( \mathcal{U}(-1, 1) \).
- \textit{Dilation}: Sampled from an exponential scale \( \kappa = \lfloor 2^{a} \rfloor \), where \( a \sim \mathcal{U}(0, \log_{2}\frac{|x|}{|w|-1}) \), \( |x| \) is the cardinality (length) of input signal and \( |w| \) is the length of kernel.
- \textit{Padding}: A binary random decision is made, with equal probability, to apply padding to the input \( x \). With padding, kernels are centered at the first and last indices of \( x \).

- \textit{Stride}: The stride value is always set to one for all kernels.

2) Features Extraction: The two most important aspects of Rocket in terms of achieving state-of-the-art accuracy are the use of dilation, sampled on an exponential scale, and the use of PPV \cite{12}. Hence, we only focus on extracting PPV features in this paper, without loss of generality.

The convolution operation \( \phi(\cdot) \) for a time series \( x \) and convolution kernel \( w \) is

\[
    f = \phi(x, w) = \left( \sum_{j=1}^{\|w\|} x_{i+j \cdot \kappa} \cdot w_j \right)_{i = 1, \ldots, |x|},
\]

where \( f \) is the extracted feature vector using dilation \( \kappa \) and \( |\cdot| \) is the cardinality. The PPV value is then computed as

\[
    k = \frac{1}{|f|} \sum_{i=1}^{\|f\|} 1[f_i > 0],
\]

where \( f_i \) is the \( i \)-th element of the vector \( f \), and \( 1[f_i > 0] \) is the indicator function such that \( 1[f_i > 0] = 1 \) if \( f_i > 0 \) and \( 1[f_i > 0] = 0 \) otherwise. Hence, for the input time series \( x_n \), the extracted feature vector using the bank of convolution kernels \( \Theta \) is \( k_n = (k_1, \ldots, k_D) \).

3) Training the Classifier: Similar to \cite{10}, a Ridge regression classifier \( \psi(k) \) is trained using the extracted features \( \{(k_1, y_1), \ldots, (k_N, y_N)\} \) corresponding to the input time series. This classifier does not require extensive hyperparameter setting, except for the regularization parameters which can be set quickly with cross-validation.

B. Optimization

S-Rocket performs structured pruning on the kernels \( \Theta \) for removing redundant and less efficient kernels.
Algorithm 1: S-Rocket

// Parameters Setup
Set $t = 0$ // Optimization counter
Set $D = 10,000$ // Number of kernels
Set $N$ // Number of training samples
Initialize $\Theta = ()$ // Empty set of kernels
Initialize $K = ()$ // Empty set of PPVs
Initialize $S^{(0)}$ // States initialization according to Section III-B1

// Kernels Initialization
for $d = 1 \rightarrow D$ // Kernel counter
    Initialize the kernel $w_d$ // Refer to Section III-A1
    Add $w_d$ to $\Theta$
end for

// Feature Extraction
for $n = 1 \rightarrow N$ do // Training sample counter
    Initialize $k_n = ()$ // Empty set of PPVs per sample
    for $d = 1 \rightarrow D$ do // Kernel counter
        Compute $k_d$ using (2)
        Add $k_d$ to $k_n$
    end for
    Add $k_n$ to $K$
end for

// Pre-Training
Train the classifier $\psi$

// Optimization
Compute loss of $S^{(0)}$ as $L^{(0)}$ using (5)
for $t = 1 \rightarrow N_{epoch}$ do // epoch counter
    for $i = 1 \rightarrow S$ do // States counter
        Generate mutually different $i_1, i_2, i_3 \in \{1, \ldots, S\}$
        Generate a random number $r_d \in [0,1]
        Compute mutation vector $v_{i,d}$ using (7)
    end for
    Select candidate state $s_i^{(t)}$ using (8)
end for

// Post-Training
Train the classifier $\psi$ using (6) and $s_i^{(t)}$

1) Initialization: Let $s_d \in \{0, 1\}$ represent the activation state of each kernel, which can be extended to the state vector $s_i^{1 \times D}$ for $D$ kernels. In the initialization step, $t = 0$, a pool of candidate state vectors $S^{(t)} \subset Z_2^{S \times D}$ is initialized such that $S$ is an even number and denotes the number of candidate state vectors, $s_i^{(t)} \sim Bernoulli(0.5)$ for $i \in (1, \ldots, S/2)$, and $s_i^{(t)} = 1$ for $i \in (S/2 + 1, \ldots, S)$ along all $d \in (1, \ldots, D)$.

2) Objective Function: The optimization objective is to minimize the number of active kernels while maximizing the classification accuracy. Hence, we need to find a best state vector $s_b$ by minimizing the objective function

$$L = \frac{1}{2} (L_D - L_A + 1),$$

where $L$ is the objective function value (OFV) and $L_D$ represents the number of active kernels, defined as

$$L_D = \sum_{d=1}^{D} s_d / D$$

and $L_A$ is the accuracy of the classifier, defined as

$$L_A = \frac{1}{N} \sum_{n=1}^{N} 1[y_n, \text{argmax}(p_{n,1}, \ldots, p_{n,C})],$$

where $C = \{1, \ldots, C\}$ is the set of target classes, $1[\cdot, \cdot]$ is the identifier function (i.e. $1[y, c] = 1$ if $y = c$ and $1[y, c] = 0$ if $y \neq c$), and

$$\psi(k_n \circ s),$$

where $\circ$ is the element-wise multiplication and $\psi(\cdot)$ is the classifier. It is obvious that since $0 \leq L_A \leq 1$, and $0 \leq L_D \leq 1$, we have $0 \leq L \leq 1$. This objective function is a trade-off between two costs so that $L_D$ increases with the number of active kernels and $(1 - L_A)$ decreases with a higher classification accuracy.

3) Searching for the Best State Vector: Since $D$ is generally a large number (e.g. $D = 10,000$ as recommended in [10]), $2^D$ possible state vectors exist. Hence, searching for a best state vector which corresponds to the minimum objective function value is an NP-hard combinatorial problem. Motivated by [22] and [58], we propose a global optimization approach to minimize $L$.

As Algorithm 1 shows, a mutation vector is computed at each optimization epoch $t$ for each candidate state vector $s_i^{(t-1)} \in S^{(t-1)}$ and all $d \in (1, \ldots, D)$ as

$$v_{i,d} = \begin{cases} 1 - s_{i,d}^{(t-1)}, & \text{if } s_{i_2,d}^{(t-1)} \neq s_{i_3,d}^{(t-1)} \& r_d < F \\
1, & \text{otherwise} \end{cases},$$

where $F$ is the mutation factor [59], $r_d \in [0,1]$ is a random number, and $i_1, i_2, i_3 \in (1, \ldots, S)$ are different indices in the pool of candidate kernels. Then, the crossover operation is defined as

$$s_{i,d}^{(t)} = \begin{cases} v_{i,d} \times s_{i,d}^{(t-1)}, & \text{if } r_d \in [0,1] \leq C_r \\
v_{i,d} \times s_{i,d}^{(t-1)}, & \text{otherwise} \end{cases},$$

where $C_r$ is the crossover coefficient [59]. The parameters $C_r$ and $F$ control exploration and exploitation of the pool of candidate state vectors on the optimization landscape. The objective function value of each state vector $s_i^{(t)}$ is then compared with its corresponding parent using (3) as

$$s_i^{(t)} = \begin{cases} s_i^{(t)}, & \text{if } L(s_i^{(t)}) \leq L(s_i^{(t-1)}) \\
\text{argmin}_{b \in (1, \ldots, S)} \{L(s_b^{(t)})\}, & \text{otherwise} \end{cases},$$

for all $i \in (1, \ldots, S)$.

In each epoch, the best state vector $s_b^{(t)}$ is selected where

$$b = \text{argmin}_{b \in (1, \ldots, S)} \{L(s_b^{(t)})\}$$

and its corresponding objective function value is

$$L(s_b^{(t)}) = \text{min} \{L(s_1^{(t)}), \ldots, L(s_S^{(t)})\},$$

where the number of active kernels (features) is $D' = |s_b^{(t)}|$ and $|\cdot|$ is the cardinality.
### Table I: Average classification performance (Avg. Acc.), average objective function value according to \( f \) (Avg. OFV), Matthews correlation coefficient (MCC), and average number of kept features (Avg. \( D' \)), scaled to \([0,1]\) for selecting kernels from Rocket. The reported values are averaged over 10 independent runs.

| Dataset        | Rocket Acc. | S-Rocket Acc. | Random \( f \) | Soft Filter \( f \) |
|----------------|-------------|---------------|----------------|---------------------|
| Adiac          | 0.78        | 0.78          | 0.69           | 0.71                |
| ArrowHead      | 0.83        | 0.83          | 0.79           | 0.82                |
| Beef           | 0.82        | 0.82          | 0.70           | 0.82                |
| BeetleFly      | 0.95        | 0.95          | 0.86           | 0.91                |
| BirdChicken    | 0.90        | 0.88 (-0.02)  | 0.78           | 0.83                |
| Car            | 0.90        | 0.87 (-0.02)  | 0.81           | 0.84                |
| CBF            | 1.00        | 1.00          | 0.85           | 0.94                |
| CinCECGT       | 0.81        | 0.80 (-0.01)  | 0.71           | 0.81                |
| ChiCon         | 0.76        | 0.73          | 0.62           | 0.73                |
| Coffee         | 1.00        | 0.99 (-0.01)  | 0.95           | 0.98                |
| Computers      | 0.77        | 0.77          | 0.69           | 0.74                |
| CricketX       | 0.83        | 0.83          | 0.81           | 0.82                |
| CricketY       | 0.85        | 0.85          | 0.78           | 0.80                |
| CricketZ       | 0.85        | 0.85          | 0.79           | 0.82                |
| DiveSZRed      | 0.95        | 0.95          | 0.88           | 0.94                |
| DisPhaOutAG    | 0.75        | 0.75          | 0.69           | 0.73                |
| DisPhaOutCor   | 0.77        | 0.77          | 0.69           | 0.74                |
| DoLoDay        | 0.65        | 0.64 (-0.01)  | 0.58           | 0.54                |
| DoLoGam        | 0.8         | 0.81 (+0.01)  | 0.75           | 0.78                |
| DoLoWKE        | 0.97        | 0.94 (-0.03)  | 0.82           | 0.92                |
| Earthquakes    | 0.75        | 0.75          | 0.69           | 0.75                |
| ECG5000        | 0.90        | 0.88 (-0.02)  | 0.81           | 0.89                |
| ECG5000        | 0.95        | 0.95          | 0.73           | 0.89                |
| ECGFiveDays    | 1.00        | 1.00          | 0.89           | 0.98                |
| EOGHSignal     | 0.57        | 0.55 (-0.02)  | 0.49           | 0.55                |
| EOGVSsignal    | 0.44        | 0.44          | 0.41           | 0.42                |
| FaceAll        | 0.79        | 0.80 (+0.01)  | 0.69           | 0.78                |
| FaceFour       | 1.00        | 1.00          | 0.82           | 0.98                |
| FacesUCR       | 0.96        | 0.96          | 0.88           | 0.91                |
| FiftyWords     | 0.85        | 0.85          | 0.76           | 0.66                |
| Average        | 0.84        | 0.83 (-0.01)  | 0.75           | 0.80                |

4) **Termination Criteria**: The optimization process terminates after \( N_{epoch} \) epochs or upon states convergence where all the states in \( S(t) \) converge to a state \( s_b \in S(t) \). This can be detected by

\[
\Delta L = L(s_b(t)) - \frac{1}{S} \sum_{j=1}^{S} L(s_j(t)),
\]

where if \( \Delta L = 0 \) the optimization process terminates \([22]\).

**C. Post-Training**

The sparse best state vector \( s_b \) represents participation status of a kernel in feature extraction, where \( s_b,d = 1 \) means kernel \( d \) remains active. The sparse features are used to retrain the Ridge regression classifier by solving a regression model where the loss function is the linear least squares function and regularization is given by the \( l_2 \)-norm.

**IV. Experiments**

**A. Data**

Similar to \([10]\), experiments are conducted on the UCR archive \([15]\) time series classification datasets (the first 30 datasets). Generally, these datasets have limited training samples with a similar/larger size test dataset.

**B. Setup**

The open-source Rocket and MiniRocket codes were used for the experiments\(^2\). The regularization coefficient of Ridge regression classifier was set based on a cross-validation search in the set of 10 numbers spaced evenly on the log scale range of \([-3, 3]\). Our developed codes for S-Rocket are available online\(^3\). In the experiments, otherwise stated, the number of random kernels is \( D = 10,000 \) \([10]\), the number of training epochs is 500, and the average results of 10 independent runs are reported. The number of candidate state vectors is \( S = 8 \) where 50% of them are initialized from the Bernoulli distribution with the probability 0.5 and the others are initialized as 1 and then shuffled.

**C. Classification Performance Analysis**

Tables \([4]\) and \([10]\) show the average classification performance (Acc.), average objective function value (OFV) according to \( f \), Matthews correlation coefficient (MCC), and ratio of kept kernels (\( D' \)) for Rocket and MiniRocket and the corresponding S-Rocket and S-MiniRocket implementation.

\(^2\)https://github.com/angus924/rocket

\(^3\)https://github.com/salehinejad/rocket
respectively. The S-Rocket model is compared with $l_1$-norm pruning \([56]\). Soft Filter pruning \([57]\), and a Random mask pruning approach. In the Random pruning approach, a random mask is applied on the kernels and the classifier is trained with the corresponding PPV values, similar to the S-Rocket setup. The pruning rate in these models is set to the $D'$ rate found by the S-Rocket model.

From the classification accuracy perspective, S-Rocket has an equal performance in 19, lower performance in 9 (\(\leq 3\%\)), and better performance in 2 (\(\leq 1\%\)) datasets. Similarly, S-MiniRocket has an equal performance in 17, lower performance in 12 (\(\leq 4\%\)), and better performance in 1 (\(\leq 1\%\)) datasets. S-Rocket and S-MiniRocket on average use about 39\% and 36\% of the kernels ($D'$), respectively, to achieve an almost similar accuracy to the original model. Hence, the OFV using \([3]\) is lower for S-Rocket and S-MiniRocket in

### Table II: Average classification performance (Avg. Acc.), average objective function value according to \([3]\) (Avg. OFV), Matthews correlation coefficient (MCC), and average number of kept features (Avg. $D'$), scaled to [0, 1] for selecting kernels from MiniRocket. The reported values are averaged over 10 independent runs.

| Dataset       | Rocket | S-Rocket | Random | $l_1$-norm [56] | Soft Filter [57] |
|---------------|--------|----------|--------|-----------------|------------------|
| Acc.          | Acc.†  | Acc.†    | Acc.†  | Acc.†           | Acc.†            |
| OFV           | OFV†   | OFV†     | OFV†   | OFV†            | OFV†             |
| MCC           | MCC†   | MCC†     | MCC†   | MCC†            | MCC†             |

| Dataset       | Rocket | S-Rocket | Random | $l_1$-norm [56] | Soft Filter [57] |
|---------------|--------|----------|--------|-----------------|------------------|
| Acc.          | Acc.†  | Acc.†    | Acc.†  | Acc.†           | Acc.†            |
| OFV           | OFV†   | OFV†     | OFV†   | OFV†            | OFV†             |
| MCC           | MCC†   | MCC†     | MCC†   | MCC†            | MCC†             |

Fig. 3: Classification accuracy vs. ratio of active convolution kernels ($D'$) for Rocket, MiniRocket, and the corresponding pruned models.

![Diagram](image.png)
Fig. 4: Convergence plots of S-Rocket and S-MiniRocket during 500 optimization epochs. OFV: The state vector with lowest objective function value according to (3) in each epoch; Acc.: The state vector with highest classification accuracy in each epoch; $D'$: The state vector with lowest number of selected convolution kernels in each epoch. The values are averaged over 10 independent runs.

Fig. 5: Average objective function value (OFV) of S-Rocket over 10 independent runs for population sizes $S \in \{4, 8, 16, 32, 64\}$.

Fig. 6: The objective function value (OFV) of S-Rocket for the ArrowHead and BeetleFly datasets averaged over 10 independent runs for $F \in \{0.3, 0.6, 0.9\}$ and $C_r \in \{0.3, 0.6, 0.9\}$.

The optimization landscape (referred to as exploration) and gradually moves toward fine-tuning of the solutions (referred to as exploitation). In some cases, the population may experience premature convergence due to lack of diversity and an early convergence to a local optima. It may also suffer from stagnation where the population remains diverse during the optimization process.

Figure 4 shows the average of accuracy, ratio of active kernels, and objective function value using $[\text{Scaled to } [0,1]]$, scaled to $[0,1]$, over 10 independent runs and 500 epochs for the ArrowHead and BeetleFly datasets. The plots show that, generally, the optimizer is increasing the accuracy while decreasing the number of active kernels and the objective function value. As an example, the exploration phase is observable before about epoch 80 in Figure 4(a).

In general, a large population size enhances exploration of the optimization landscape and decreases the convergence rate. A small population size helps faster convergence but increases the risk of trapping in a local optima and premature convergence.

D. Convergence Analysis

Population-based optimization algorithms perform a global search and typically converge to a local solution. Generally, the algorithm spends a number of early epochs for exploring comparison with their original counterpart. Overall, the models with pruned kernels have achieved almost similar performance to the original model with less than 40% of the kernels. Figure 5 shows the trade-off between accuracy and the number of selected features (kernels) for all datasets in Tables I and II. Since the ratio of utilized kernels for Rocket and MiniRocket is $D' = 1$, the corresponding indicators have a linear pattern. The center of each cluster represents the average accuracy and the average number of kept kernels over all datasets.
$S \in \{4, 8\}$. Particularly, $S = 64$ has a very slow convergence rate.

After about epoch 180, the $S \in \{4, 8, 16\}$ are transiting from exploration to exploitation and a slow improvement in the OFV is observable. The small populations $S \in \{4, 8\}$ are relatively closer values but the $S = 8$ has a better balance of exploration and exploitation and can achieve a lower OFV. A similar behaviour is observable for the BeetleFly dataset.

The mutation factor $F$ in (7) and the cross-over rate $C_r$ in (8) are other parameters to control the diversity of search and convergence rate (59). A larger mutation factor $0 \leq F \leq 1$ increases the probability of flipping a state in (7). Similarly, a larger cross-over rate $0 \leq C_r \leq 1$ increases the chance of using a mutated vector and smaller value increases the chance of inheritance from the previous generation. Figure 3 shows the objective function values of S-Rocket for the ArrowHead and BeetleFly datasets averaged over 10 independent runs for $S = 8$, $F \in \{0.3, 0.6, 0.9\}$, and $C_r \in \{0.3, 0.6, 0.9\}$. These plots show that a combination of larger mutation factor and cross-over rates (i.e. $C_r = 0.9$ and $F = 0.9$) significantly accelerate convergence of the optimizer to a lower objective function value.
E. Monte Carlo Simulations

S-Rocket reduces complexity of Rocket by pruning less important convolution kernels while maintaining the classification accuracy. Hence, there is a trade-off between these two objectives. Figure 7 shows a visualization of S-Rocket and S-MiniRocket for the ArrowHead and BeetleFly datasets. A desired area is where the accuracy is maximized while the number of input features is minimized (in green). The S-Rocket/S-MiniRocket OFV marker represents the best model during optimization with respect to the accuracy and the number of active kernels using $\mathbf{3}$. The S-Rocket/S-MiniRocket Acc. marker represents the best model during optimization with respect to maximizing the accuracy. The S-Rocket/S-MiniRocket $D'$ marker represents the best model during optimization with respect to minimizing the number of kernels.

Figure 8 shows Monte Carlo simulation results of classification accuracy values for different arbitrary ratios of active kernels ($D'$). For each $D'$, 1,000 random state vectors (without replacement) from a standard uniform distribution are generated. Then, each vector $s_i$ is applied to the extracted features from random kernels of a trained Rocket model as $s_i \odot k_n$, where $k_n$ is extracted from the test dataset. These figures show efficiency of the S-Rocket in finding a state vector which maximizes the accuracy while minimizing the number of active kernels. For instance in Figure 8(c), the accuracy of S-Rocket is 82% and 16% active kernels (green +), which is equivalent to the Rocket at full capacity (i.e. 100% active kernels).

F. Feature Space Analysis

Principal components analysis (PCA) [62] and t-distributed stochastic neighbor embedding (t-SNE) [65] are well-established methods for studying and visualization of features in a lower dimensional space such as in human activity recognition [64] and medical images analysis [65].

Figures 9 and 10 show the extracted features from random convolution kernels after reduction and normalization to a two-dimensional space using PCA for the Rocket, S-Rocket, MiniRocket, and S-MiniRocket models using Adiac, ArrowHead, Beef, and BeetleFly datasets. These scatter plots...
represent the two-dimensional features of samples in each dataset, where the corresponding time series class is color-coded. For example in Figure 9, the ArrowHead dataset has three classes which are denoted by green, red, and blue. Figures 9 shows the density of features in a lower dimensional space for Rocket and S-Rocket. The shorter the distance between S-Rocket and Rocket feature clusters per class (color coded) show closer feature extraction performance of S-Rocket to Rocket.

In other words, it shows that S-Rocket can remote redundant and less important kernels (and hence the corresponding features), without affecting the important ones. This also supports the very close classification performance results of S-Rocket to Rocket in Table 1. A similar pattern is observable in Figure 10 for the features extracted using MiniRocket and S-MiniRocket.

The normalized Euclidean distance between extracted features using Rocket and S-Rocket from the time series samples studied in Figure 9 is plotted in Figure 11. These plots show that the features after pruning less important kernels are close to the features before prunning. Figure 11 also shows a similar pattern for MiniRocket and S-MiniRocket.

G. Complexity Analysis

1) Training: The Ridge regression classifier has a complexity of \(O_R = O(N^2 \cdot D)\) when \(N < D\). The implementation of the transforms in Rocket has a computational complexity of \(O_T = O(D \cdot N \cdot l_{input})\), where \(l_{input}\) is the length of the time series. Hence, the complexity of Rocket is a linear function of the number of features (kernels) and its total complexity is

\[
O_{\text{Rocket}}^{(\text{training})} = O(D \cdot N \cdot l_{input}) + O(N^2 \cdot D). \tag{13}
\]

Since S-Rocket has three steps in the training phase, its complexity is

\[
O_{S\text{-Rocket}} = O_{\text{Pre}} + O_{\text{Opt}} + O_{\text{Post}}, \tag{14}
\]

where \(O_{\text{Pre}} = O_{\text{Rocket}}\) is for the pre-training and \(O_{\text{Opt}} = O(S \cdot N^3 \cdot D \cdot N_{epochs})\) for the optimization, with the worst-case assumption of \(D' = D\). The complexity of the post-training step is

\[
O_{\text{Post}} = O(D' \cdot N \cdot l_{input}) + O(N^2 \cdot D'), \tag{15}
\]

where \(D' \leq D\). Therefore the total training cost for S-Rocket is

\[
O_{S\text{-Rocket}}^{(\text{training})} = O_{\text{Rocket}}^{(\text{training})} + O(S \cdot N^3 \cdot D \cdot N_{epochs}) + O(D' \cdot N \cdot l_{input}) + O(N^2 \cdot D'), \tag{16}
\]

which can be simplified to

\[
O_{S\text{-Rocket}}^{(\text{training})} = O_{\text{Rocket}}^{(\text{training})} + O(S \cdot N^3 \cdot D \cdot N_{epochs}). \tag{17}
\]

2) Inference: In inference, the complexity of Rocket and S-Rocket for a single time series is

\[
O_{\text{Rocket}}^{(\text{inference})} = O(D \cdot l_{input}) + O(D), \tag{18}
\]

and

\[
O_{S\text{-Rocket}}^{(\text{inference})} = O(D' \cdot l_{input}) + O(D'), \tag{19}
\]

respectively. Since \(D' \leq D\), then \(O_{S\text{-Rocket}}^{(\text{inference})} \leq O_{\text{Rocket}}^{(\text{inference})}\).

Table IV presents the execution time of different steps of the Rocket and S-Rocket, which shows the inference time of S-Rocket is less than Rocket in all experiments.

| Dataset   | Common Steps (in Seconds) | Rocket (in Seconds) | S-Rocket (in Seconds) |
|-----------|---------------------------|--------------------|-----------------------|
|           | Initialization & Convolution | Training/Pre-Training | Inference | Optimization (per epoch) | Post-Training | Inference |
| Adiac     | 4.020                     | 5.320              | 0.806                | 1.067                  | 4.952         | 0.660     |
| ArrowHead | 1.693                     | 0.211              | 0.050                | 0.027                  | 0.113         | 0.008     |
| Beef      | 1.401                     | 0.108              | 0.023                | 0.029                  | 0.081         | 0.015     |
| BeetleFly | 0.893                     | 0.066              | 0.034                | 0.013                  | 0.038         | 0.004     |

V. CONCLUSIONS

Training a linear classifier using feature generated from a bank of random convolution kernels (without training the kernels) is a fast and efficient approach for time series classification. Rocket and MiniRocket are two methods based on this idea for time series classification. In this paper, we propose a method for pruning less efficient and redundant kernels in Rocket and MiniRocket while maintaining the classification accuracy of the original models, called S-Rocket and S-MiniRocket, respectively. This approach can reduce computational complexity of Rocket and MiniRocket in inference mode for implementation on devices with limited resources such as edge devices. The results show that S-Rocket and S-MiniRocket can prune up to 99% of the random convolution kernels in some standard datasets without noticeable reduction of the classification performance. Our analysis in the feature space shows that the extracted features before and after pruning of the convolution kernels are very similar, which supports the efficiency of the proposed approach in removing unnecessary kernels without affecting the performance of classifier.

VI. ACKNOWLEDGMENT

This work was partially supported by the Mobile AI Lab established between Huawei Technologies Co. LTD Canada and The Governing Council of the University of Toronto.

REFERENCES

[1] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, “Deep learning for time series classification: a review,” Data mining and knowledge discovery, vol. 33, no. 4, pp. 917–963, 2019.

[2] S. Yousefi, H. Narui, S. Dayal, S. Ermon, and S. Valaee, “A survey on behavior recognition using wifi channel state information,” IEEE Communications Magazine, vol. 55, no. 10, pp. 98–104, 2017.
[3] E. F. d. S. Soares, H. Salehinejad, C. A. V. Campos, and S. Valaee, “Recurrent neural networks for online travel mode detection,” in 2019 IEEE Global Communications Conference (GLOBECOM). IEEE, 2019, pp. 1–6.

[4] J. Hirschberg and C. D. Manning, “Advances in natural language processing,” Science, vol. 349, no. 6245, pp. 261–266, 2015.

[5] Z. Che, S. S. Eshghi, K. Cho, D. Sontag, and Y. Liu, “Recurrent neural networks for multivariate time series with missing values,” Scientific reports, vol. 8, no. 1, pp. 1–12, 2018.

[6] H. Salehinejad and S. Rahnamayan, “Customer shopping pattern prediction: A recurrent neural network approach,” in Computational Intelligence (SSCI), 2016 IEEE Symposium Series on. IEEE, 2016, pp. 1–5.

[7] A. Rajkomar, E. Oren, K. Chen, A. M. Dai, N. Hajaj, M. Hardt, P. J. Liu, X. Liu, J. Marcus, M. Sun et al., “Scalable and accurate deep learning with electronic health records,” NPJ Digital Medicine, vol. 1, no. 1, pp. 1–10, 2018.

[8] H. Salehinejad, H. Ho, H.-M. Lin, P. Crivellaro, O. Samarodin, M. T. Arciniegas, Z. Merali, S. Suthiphosuwan, A. Bharatha, K. Ytom et al., “Deep sequential learning for cervical spine fracture detection on computed tomography imaging,” in 2021 IEEE 18th International Symposium on Biomedical Imaging (ISBI). IEEE, 2021, pp. 1911–1914.

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

[10] A. Dempster, F. Petitjean, and G. I. Webb, “Rocket: exceptionally fast and accurate time series classification using random convolutional kernels,” Data Mining and Knowledge Discovery, vol. 34, no. 5, pp. 1454–1495, 2020.

[11] T. Xu, H. Salehinejad, Q. Huang, E. Cho, S. Valaee, and C.-W. Chow, “Late breaking abstract - detection of small airway obstruction patterns in forced oscillometry(fo) and spirometry using artificial neural networks,” European Respiratory Journal, vol. 52, no. supp 62, 2018.

[12] A. Dempster, D. F. Schmidt, and G. I. Webb, “Minirocket: A very fast (almost) deterministic transform for time series classification,” arXiv preprint arXiv:2012.08791, 2020.

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

[14] B. Lucas, A. Shifaz, C. Pelletier, L. O’Neill, N. Zaidi, B. Goethals, F. Petitjean, and G. I. Webb, “Proximity forest: an effective and scalable distance-based classifier for time series,” Data Mining and Knowledge Discovery, vol. 33, no. 3, pp. 607–635, 2019.

[15] C. A. Dax, A. Bagnall, K. Kamgar, C.-C.-M. Yeh, Y. Zhu, S. Gharghabi, C. A. Ratnamahatana, and E. Keogh, “The ucr time series archive,” IEEE/CAA Journal of Automatica Sinica, vol. 6, no. 6, pp. 1293–1305, 2019.

[16] C. Lundy and J. M. O’Toole, “Random convolution kernels with multi-scale decomposition for preterm eeg inter-hurt detection,” arXiv preprint arXiv:2108.02039, 2021.

[17] C. W. Tan, M. Salehi, and G. Mackellar, “Detecting driver’s distraction using long-term recurrent convolutional network,” arXiv preprint arXiv:2004.11839, 2020.

[18] A. R. Andreu-Perez, M. Kiani, J. Andreu-Perez, P. Reddy, J. Andreu-Abela, M. Pinto, and K. Izzetoglu, “Single-trial recognition of video gamer’s expertise from brain haemodynamic and facial emotion responses,” Brain Sciences, vol. 11, no. 1, p. 106, 2021.

[19] H. Salehinejad and S. Valaee, “Litehar: Lightweight human activity recognition from wifi signals with random convolution kernels,” arXiv preprint arXiv:2201.09310, 2022.

[20] S. Oymak and M. Soltanolkotabi, “Toward moderate overparameterization: Global convergence guarantees for training shallow neural networks,” IEEE Journal on Selected Areas in Information Theory, vol. 1, no. 1, pp. 84–105, 2020.

[21] H. Salehinejad, “Energy models for pruning neural networks,” Ph.D. dissertation, University of Toronto (Canada), 2021.

[22] H. Salehinejad and S. Valaee, “Edropout: Energy-based dropout and pruning of deep neural networks,” IEEE Transactions on Neural Networks and Learning Systems, 2021.

[23] K. Dabiri, M. Maleknomamadi, A. Sheikholeslami, and H. Tamura, “Random exchange hardware with automatic temperature selection and parallel trial,” IEEE Transactions on Parallel and Distributed Systems, vol. 31, no. 7, pp. 1681–1692, 2020.

[24] K. V. Price, “Differential evolution,” pp. 187–214, 2013.
[51] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, “What is the best multi-stage architecture for object recognition?” in 2009 IEEE 12th international conference on computer vision. IEEE, 2009, pp. 2146–2153.

[52] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, “How transferable are features in deep neural networks?” arXiv preprint arXiv:1411.1792, 2014.

[53] A. Rahimi and B. Recht, “Weighted sums of random kitchen sinks: replacing minimization with randomization in learning.” in Nips. Citeseer, 2008, pp. 1313–1320.

[54] A. Morrow, V. Shankar, J. Petersohn, A. Joseph, B. Recht, and N. Yosef, “Convolutional kitchen sinks for transcription factor binding site prediction,” arXiv preprint arXiv:1706.00125, 2017.

[55] S. Bai, J. Z. Kolter, and V. Koltun, “An empirical evaluation of generic convolutional and recurrent networks for sequence modeling,” arXiv preprint arXiv:1803.01271, 2018.

[56] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, “Pruning filters for efficient convnets.” ICLR, 2016.

[57] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang, “Soft filter pruning for accelerating deep convolutional neural networks.” Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, 2018.

[58] H. Salehinejad and S. Valaee, “A framework for pruning deep neural networks using energy-based models,” in ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2021, pp. 3920–3924.

[59] H. Salehinejad, S. Rahnamayan, and H. R. Tizhoosh, “Micro-differential evolution: Diversity enhancement and a comparative study.” Applied Soft Computing, vol. 52, pp. 812–833, 2017.

[60] J. Lampinen, I. Zelinka et al., “On stagnation of the differential evolution algorithm.” in Proceedings of MENDEL, 2000, pp. 76–83.

[61] H. Salehinejad, S. Rahnamayan, H. R. Tizhoosh, and S. Y. Chen, “Micro-differential evolution with vectorized random mutation factor,” in 2014 IEEE congress on evolutionary computation (CEC). IEEE, 2014, pp. 2055–2062.

[62] I. Jolliffe, “Principal component analysis,” Encyclopedia of statistics in behavioral science, 2005.

[63] L. Van der Maaten and G. Hinton, “Visualizing data using t-sne.” Journal of machine learning research, vol. 9, no. 11, 2008.

[64] Y. Zhang, Y. Zheng, Q. Kian, G. Zhang, Y. Liu, C. Wu, and Z. Yang, “Widar3. 0: Zero-effort cross-domain gesture recognition with wi-fi.” IEEE Transactions on Pattern Analysis and Machine Intelligence, 2021.

[65] H. Salehinejad, E. Colak, T. Dowdell, J. Barfett, and S. Valaee, “Synthesizing chest x-ray pathology for training deep convolutional neural networks,” IEEE transactions on medical imaging, vol. 38, no. 5, pp. 1197–1206, 2018.

[66] J. Dongarra, M. Gates, A. Haidar, J. Kurzak, P. Luszczek, S. Tomov, and I. Yamazaki, “The singular value decomposition: Anatomy of optimizing an algorithm for extreme scale,” SIAM review, vol. 60, no. 4, pp. 808–865, 2018.

[67] X. Zhong, M. Duan, X. Zhang, and P. Cheng, “A hybrid differential evolution based on gaining-sharing knowledge algorithm and harris hawks optimization.” Plos one, vol. 16, no. 4, p. e0250951, 2021.