Data Mining and Knowledge Discovery (2020) 34:1454–1495
https://doi.org/10.1007/s10618-020-00701-z

ROCKET: exceptionally fast and accurate time series classification using random convolutional kernels

Angus Dempster[1] · François Petitjean[1] · Geoffrey I. Webb[1]

Received: 28 October 2019 / Accepted: 18 June 2020 / Published online: 13 July 2020

Abstract
Most methods for time series classification that attain state-of-the-art accuracy have high computational complexity, requiring significant training time even for smaller datasets, and are intractable for larger datasets. Additionally, many existing methods focus on a single type of feature such as shape or frequency. Building on the recent success of convolutional neural networks for time series classification, we show that simple linear classifiers using random convolutional kernels achieve state-of-the-art accuracy with a fraction of the computational expense of existing methods. Using this method, it is possible to train and test a classifier on all 85 'bake off' datasets in the UCR archive in < 2 h, and it is possible to train a classifier on a large dataset of more than one million time series in approximately 1 h.

Keywords Scalable · Time series classification · Random · Convolution

1 Introduction

Most methods for time series classification that attain state-of-the-art accuracy have high computational complexity, requiring significant training time even for smaller datasets, and simply do not scale to large datasets. This has motivated the development of more scalable methods such as Proximity Forest (Lucas et al. 2019), TS-CHIEF

Responsible editor: Aristides Gionis, Carlotta Domeniconi, Eyke Hüllermeier, Ira Assent.

✉ Angus Dempster
angus.dempster1@monash.edu

François Petitjean
francois.petitjean@monash.edu

Geoffrey I. Webb
geoff.webb@monash.edu

1 Faculty of Information Technology, Monash University, Melbourne, Australia

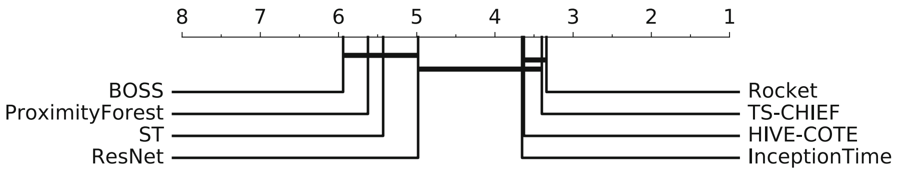


Fig. 1 Mean rank of ROCKET versus state-of-the-art classifiers on the 85 'bake off' datasets

(Shifaz et al. 2020), InceptionTime (Ismail Fawaz et al. 2019c), MrSEQL (Le Nguyen et al. 2019), and cBOSS (Middlehurst et al. 2019).

We show that state-of-the-art classification accuracy can be achieved using a fraction of the time required by even these recent, more scalable methods, by transforming time series using random convolutional kernels, and using the transformed features to train a linear classifier. We call this method ROCKET (for **R**and**O**m **C**onvolutional **KE**rnel **T**ransform).

Existing methods for time series classification typically focus on a single representation such as shape, frequency, or variance. Convolutional kernels constitute a single mechanism which can capture many of the features which have each previously required their own specialised techniques, and have been shown to be effective in convolutional neural networks for time series classification such as ResNet (Wang et al. 2017; Ismail Fawaz et al. 2019a), and InceptionTime.

In contrast to learned convolutional kernels as used in typical convolutional neural networks, we show that it is effective to generate a large number of random convolutional kernels which, in combination, capture features relevant for time series classification—even though, in isolation, a single random convolutional kernel may only very approximately capture a relevant feature in a given time series.

ROCKET achieves state-of-the-art classification accuracy on the datasets in the UCR archive (Dau et al. 2019), but requires only a fraction of the training time of existing methods. Figure 1 shows the mean rank of ROCKET versus several state-of-the-art methods for time series classification on the 85 'bake off' datasets from the UCR archive (Dau et al. 2019; Bagnall et al. 2017). (The different subsets of the UCR archive are discussed in Sect. 4.1.1, below) Restricted to a single CPU core, the total training time for ROCKET is:

- 6 min 33 s for the 'bake off' dataset with the largest training set (*ElectricDevices*, with 8926 training examples), compared to 31 min for MrSEQL, 1 h 35 min for Proximity Forest, 2 h 24 min for TS-CHIEF, 3 h 6 min for cBOSS, and 7 h 46 min for InceptionTime (trained on GPUs); and
- 4 min 18 s for the 'bake off' dataset with the longest time series (*HandOutlines*, with time series of length 2709), compared to 42 min for cBOSS, 1 h 55 min for MrSEQL, 8 h 10 min for InceptionTime (trained on GPUs), almost 3 days for Proximity Forest, and more than 4 days for TS-CHIEF.

The total compute time (training and test) for ROCKET on all 85 'bake off' datasets is 1 h 40 min, compared to 19 h 33 min for cBOSS, approximately 1 day for MrSEQL, more than 6 days for InceptionTime (trained and tested using GPUs), and more than 11 days for each of Proximity Forest and TS-CHIEF. Timings for ROCKET are averages over 10 runs, performed on a cluster using a mixture of Intel Xeon E5-2680 v3 and Intel Xeon Gold 6150 processors, restricted to a single CPU core per dataset per run. These figures represent the total compute time required for all 85 'bake off' datasets in sequence (not in parallel).

Rocket is also more scalable for large datasets, with training complexity linear in both time series length and the number of training examples. Rocket can learn from 1 million time series in 1 h 3 min, to a similar accuracy as Proximity Forest, which requires more than 16 h to train on the same quantity of data. A restricted variant of Rocket can learn from the same 1 million time series in less than 1 min, or approximately 100 times faster again, albeit to a slightly lower accuracy. Rocket is naturally parallel, and can be made even faster by using multiple CPU cores (our implementation automatically parallelises the transform across multiple CPU cores where available) or GPUs.

The rest of this paper is structured as follows. In Sect. 2, we review relevant related work. In Sect. 3, we explain Rocket in detail. In Sect. 4, we present our experimental results, including a comparison of the accuracy of Rocket against existing state-of-the-art classifiers on the datasets in the UCR archive, a scalability study, and a sensitivity analysis.

2 Related work

2.1 State-of-the-art methods

The task of time series classification can be thought of as involving learning or detecting signals or patterns within time series associated with relevant classes. '[D]ifferent problems require different representations' (Bagnall et al. 2017, p. 647), and classes may be distinguished by multiple types of patterns: 'discriminatory features in multiple domains' (Bagnall et al. 2017, p. 645).

Different methods for time series classification represent different approaches for extracting useful features from time series (Bagnall et al. 2017). Existing approaches typically focus on a single type of feature, such as frequency or variance of the signal, or the presence of discriminative subseries (shapelets). Bagnall et al. (2017) identified COTE, since superseded by HIVE-COTE (Lines et al. 2018), Shapelet Transform (Hills et al. 2014; Bostrom and Bagnall 2015), and BOSS (Schäfer 2015) as the three most accurate classifiers on the UCR archive.

BOSS is one of several dictionary-based methods which use a representation based on the frequency of occurrence of patterns in time series (Bagnall et al. 2017). BOSS has a training complexity quadratic in both the number of training examples and time series length, $O(n^2 \cdot l^2)$. cBOSS is a recent, more scalable variant of BOSS. Another related method, WEASEL, is more accurate than BOSS, but with a similar training complexity and high memory complexity (Schäfer and Leser 2017; see also Lucas et al. 2019).

Shapelet Transform is one of several methods based on finding discriminative subseries, so-called 'shapelets' (Bagnall et al. 2017). Shapelet Transform has a training complexity quadratic in the number of training examples, and quartic in time series length, $O(n^2 \cdot l^4)$. There are other, more scalable, shapelet methods, but these are less accurate (Bagnall et al. 2017).

HIVE-COTE is a large ensemble of other classifiers, including BOSS and Shapelet Transform, as well as classifiers based on elastic distance measures and frequency representations. Since Lines et al. (2018), HIVE-COTE has been considered the most accurate method for time series classification. The training complexity of HIVE-COTE is bound by the complexity of Shapelet Transform, $O(n^2 \cdot l^4)$, but its other components also have high computational complexity, such as the Elastic Ensemble with $O(n^2 \cdot l^2)$.

2.2 More scalable methods

The high computational complexity of existing state-of-the-art methods for time series classification makes these methods slow, even for smaller datasets, and intractable for large datasets. This has motivated the development of more scalable methods, including Proximity Forest, TS-CHIEF, InceptionTime, and others.

Proximity Forest is an ensemble of decision trees, using elastic distance measures as splitting criteria (Lucas et al. 2019), with a training complexity quasilinear in the number of training examples, but quadratic in time series length.

TS-CHIEF builds on Proximity Forest, incorporating dictionary-based and interval-based splitting criteria (Shifaz et al. 2020). Like Proximity Forest, TS-CHIEF has a training complexity quasilinear in the number of training examples, but quadratic in time series length.

As well as cBOSS, other recent more scalable methods include MrSEQL, MiSTiCl (Raza and Kramer 2019), and catch22 (Lubba et al. 2019). MrSEQL and MiSTiCl both use underlying representations similar to those used in many dictionary methods such as BOSS. MrSEQL uses a specialised sequence classification algorithm, MiSTiCl is based on the use of a string mining algorithm, and catch22 is a transform based on 22 predefined time series features. MrSEQL is stated to have a training complexity linear in training set size and quasilinear in time series length, while MiSTiCl is linear in both training set size and time series length. catch22 (transform only) is linear in training set size and 'near linear' in time series length.

Several methods for time series classification using convolutional neural networks have been proposed (see generally Ismail Fawaz et al. 2019a). More recently, InceptionTime (Ismail Fawaz et al. 2019c), an ensemble of five deep convolutional neural networks based on the Inception architecture, has been demonstrated to be competitive with HIVE-COTE on the UCR archive.

Convolutional neural networks are typically trained using stochastic gradient descent or closely related algorithms such as, for example, Adam (Kingma and Ba 2015). The training complexity of stochastic gradient descent is essentially linear with respect to the number of training examples, and training can be parallelised using GPUs (Goodfellow et al. 2016, pp. 147–149; Bottou et al. 2018).

2.3 Convolutional neural networks and convolutional kernels

Ismail Fawaz et al. (2019c, pp. 2–3) observe that the success of convolutional neural networks for image classification suggests that they should also be effective for time series classification, given that time series have essentially the same topology as images, with one less dimension (see also Bengio et al. 2013, pp. 1820–1821).

Convolutional neural networks represent a different approach to time series classification than many other methods. Rather than approaching the problem with a preconceived representation, convolutional neural networks use convolutional kernels to detect patterns in the input. In learning the weights of the kernels, a convolutional neural network learns the features in time series that are associated with different classes (Ismail Fawaz et al. 2019a).

A kernel is convolved with an input time series through a sliding dot product operation, to produce a feature map which is, in turn, used as the basis for classification (see Ismail Fawaz et al. 2019a). The basic parameters of a kernel are its size (length), weights, bias, dilation, and padding (see generally Goodfellow et al. 2016, ch. 9). A kernel has the same structure as the input, but is typically much smaller. For time series, a kernel is a vector of weights, with a bias term which is added to the result of the convolution operation between an input time series and the weights of the given kernel. Dilation ‘spreads’ a kernel over the input such that with a dilation of two, for example, the weights in a kernel are convolved with every second element of an input time series (see Bai et al. 2018). Padding involves appending values (typically zero) to the start and end of input time series, typically such that the ‘middle’ weight of a given kernel aligns with the first element of an input time series at the start of the convolution operation.

Convolutional kernels can capture many of the types of features used in other methods. Kernels can capture basic patterns or shapes in time series, similar to shapelets: the convolution operation will produce large output values where the kernel matches the input (see also Sect. 2.5). Further, dilation allows kernels to capture the same pattern at different scales (Yu and Koltun 2016). Multiple kernels in combination can capture complex patterns.

The feature maps produced in applying a kernel to a time series reflect the extent to which the pattern represented by the kernel is present in the time series. In a sense, this is not unlike dictionary methods, which are based on the frequency of occurrence of patterns in time series.

The kernels learned in convolutional neural networks often include filters for frequency (see, e.g., Krizhevsky et al. 2012; Yosinski et al. 2014; Zeiler and Fergus 2014). Saxe et al. (2011) demonstrate that even random kernels are frequency selective. Frequency information is also captured through dilation: larger dilations correspond to lower frequencies, smaller dilations to higher frequencies.

Kernels can detect patterns in time series despite warping. Pooling mechanisms make kernels invariant to the position of patterns in time series. Dilation allows kernels with similar weights to capture patterns at different scales, i.e., despite rescaling. Multiple kernels with different dilations can, in combination, capture discriminative patterns despite complex warping.

The success of convolutional neural networks for time series classification, such as ResNet and InceptionTime, demonstrates the effectiveness of convolutional kernels as the basis for time series classification.

2.4 Random convolutional kernels

The weights of convolutional kernels are typically learned. However, it is well established that random convolutional kernels can be effective (Jarrett et al. 2009; Pinto et al. 2009; Saxe et al. 2011; Cox and Pinto 2011).

Ismail Fawaz et al. (2019c) observe that individual convolutional neural networks exhibit high variance in classification accuracy on the UCR archive, motivating the use of ensembles of such architectures with a large number and variety of kernels (see Ismail Fawaz et al. 2019b). It may be that learning 'good' kernels is difficult on small datasets. Random convolutional kernels may have an advantage in this context (see Jarrett et al. 2009; Yosinski et al. 2014).

The idea of using convolutional kernels as a transform, and using the transformed features as the input to another classifier is well established (see, e.g., Bengio et al. 2013, p. 1803). Franceschi et al. (2019) present a method for unsupervised learning of convolutional kernels for a feature transform for time series input, based on a multilayer convolutional architecture with dilation increasing exponentially in each successive layer. The method is demonstrated using the output features as the input for a support vector machine. Random convolutional kernels have been used as the basis of feature transformations. In Saxe et al. (2011), random convolutional layers are used as the basis of a feature transform (for images), used as the input for a support vector machine.

Here, there is a link between using random convolutional kernels as a transform for time series and work in relation to random transforms for kernel methods (as in support vector machines, not to be confused with convolutional kernels). Rahimi and Recht (2008) proposed a random transform for approximating kernels for kernel methods (see also Rahimi and Recht 2009). Morrow et al. (2016) propose a method for approximating a string kernel for DNA sequences, based on Rahimi and Recht (2008), which involves transforming input sequences using random convolutional kernels, and using the resulting features to train a linear classifier. Morrow et al. (2016, p. 1) describe their method as 'a 1 layer random convolutional neural network'. Also following Rahimi and Recht (2008), Jimenez and Raj (2019) propose a similar method for approximating a cross-correlation kernel for measuring similarity between time series, involving convolving input time series with random time series of the same length to produce what they call 'random convolutional features', which can be used to train a linear classifier. Jimenez and Raj (2019) evaluate their method on a selection of binary classification datasets from the UCR archive. Farahmand et al. (2017) propose a feature transformation based on convolving input time series with random autoregressive filters. However, there are key differences between ROCKET and other methods using random convolutional kernels in terms of: (a) the configuration of the convolutional kernels (bias, length, dilation, and padding); (b) the use of nonlinearities; (c) pooling; and (d) the need with other methods to 'tune' certain hyperparameters.

Rocket uses random kernel length, dilation, and padding. We demonstrate that kernel dilation, in particular, is of critical importance to the high accuracy achieved by Rocket (see Sect. 4.3.4). Morrow et al. (2016) use a fixed kernel length and Jimenez and Raj (2019) use kernels of the same length as the input. Neither use kernel dilation or any variety in terms of padding. Rocket does not use any nonlinearities, and uses both global max pooling as well as a very different form of pooling, that is, the proportion of positive values or *ppv* (see Sect. 3.2). We demonstrate that the use of *ppv* has the single largest effect on accuracy of any aspect of Rocket (see Sect. 4.3.6). In contrast, Morrow et al. (2016) and Jimenez and Raj (2019) both use cosine nonlinearities and a kind of global average pooling. Rocket also uses bias differently. For Rocket, there is a close connection between bias and *ppv*, where bias acts as a kind of 'threshold' for *ppv* (see Sect. 3.2). Additionally, the only hyperparameter for Rocket is the number of convolutional kernels, k. In practice, this is left at its default value of 10,000, and does not require tuning. Both Morrow et al. (2016) and Jimenez and Raj (2019) tune different hyperparameters via cross-validation.

2.5 Shapelets and random shapelets

There are also similarities with methods based on shapelets. Broadly, both methods using convolutional kernels and methods using shapelets attempt to discriminate between classes based on the similarity of input time series to a set of patterns. The convolution operation is based on the dot product which, if used in conjunction with global max pooling, is very similar to Euclidean distance typically used in shapelet methods (Cui et al. 2016). On this basis, there is a connection between methods using random convolutional kernels and methods using random shapelets such as Wistuba et al. (2015), Renard et al. (2015), and Karlsson et al. (2016). However, there are several important characteristics differentiating Rocket from shapelet methods, including methods using random shapelets.

The equivalence between shapelet distance and the convolution operation does not necessarily hold for pooling methods other than global max pooling and, as noted above, in addition to global max pooling, Rocket uses a very different kind of pooling, i.e., *ppv* (see Sect. 3.2). There is no direct equivalent for some aspects of convolutional kernels in shapelet methods, such as dilation and bias. Again, the use of dilation and *ppv* are the two most important aspects of Rocket (see Sects. 4.3.4, 4.3.6). Bias, in turn, is integral to *ppv*, acting as a kind of threshold value. As in Wistuba et al. (2015), Renard et al. (2015), and Karlsson et al. (2016), shapelets are typically sampled from the input, and are typically longer subsequences (up to the length of the input). The convolutional kernels used in Rocket are short, and are not sampled from the input.

3 Method

Rocket transforms time series using a large number of random convolutional kernels, i.e., kernels with random length, weights, bias, dilation, and padding. The transformed features are used to train a linear classifier. For all but the largest datasets, we use a ridge regression classifier, which has the advantage of fast cross-validation for the regularisation hyperparameter (and no other hyperparameters). Nonetheless, as logistic regression trained using stochastic gradient descent is more scalable for very large datasets, we use logistic regression when the number of training examples is substantially greater than the number of features.

Four things distinguish ROCKET from convolutional layers as used in typical convolutional neural networks, and from previous work using convolutional kernels (including random kernels) with time series:

1. ROCKET uses a very large number of kernels. As there is only a single 'layer' of kernels, and as the kernel weights are not learned, the cost of computing the convolutions is low, and it is possible to use a very large number of kernels with relatively little computational expense.
2. ROCKET uses a massive variety of kernels. In contrast to typical convolutional neural networks, where it is common for groups of kernels to share the same size, dilation, and padding, for ROCKET each kernel has random length, dilation, and padding, as well as random weights and bias.
3. In particular, ROCKET makes key use of kernel dilation. In contrast to the typical use of dilation in convolutional neural networks, where dilation increases exponentially with depth (e.g., Yu and Koltun 2016; Bai et al. 2018; Franceschi et al. 2019), we sample dilation randomly for each kernel, producing a huge variety of kernel dilation, capturing patterns at different frequencies and scales, which is critical to the performance of the method (see Sect. 4.3.4).
4. As well as using the maximum value of the resulting feature maps (broadly speaking, similar to global max pooling), ROCKET uses an additional and, to our knowledge, novel feature: the proportion of positive values (or *ppv*). This enables a classifier to weight the prevalence of a given pattern within a time series. This is the single element of the ROCKET architecture that is most critical to its outstanding accuracy (see Sect. 4.3.6).

Further, while the combination of ROCKET and logistic regression forms, in effect, a single-layer convolutional neural network with random kernel weights, where the transformed features form the input for a trained softmax layer, there are important differences between ROCKET and other neural network architectures. In particular: (a) ROCKET does not use a hidden layer, or any nonlinearities; (b) the features produced by ROCKET are independent of each other (there are no 'connections' between the convolutional kernels); and (c) strictly speaking, ROCKET does not mandate the use of a particular classifier and, indeed, we suggest using different classifiers (namely, the ridge regression classifier or logistic regression) in different contexts.

We implement ROCKET in Python, using just-in-time compilation via Numba (Lam et al. 2015). For the experiments on the datasets in the UCR archive, we use a ridge regression classifier from scikit-learn (Pedregosa et al. 2011). For the experiments studying scalability, we integrate ROCKET with logistic regression and Adam, implemented using PyTorch (Paszke et al. 2017). Our code will be made available at https://github.com/angus924/rocket.

In developing ROCKET, we have endeavoured to not overfit the architecture and its parameters to the entire UCR archive (see Bagnall et al. 2017, p. 608). At the same time, in order to develop the method, we required representative time series datasets. Accordingly, we chose to develop the method on a subset of 40 randomly-selected datasets from the 85 'bake off' datasets. We refer to these as the 'development' datasets, and the remaining 45 datasets as the 'holdout' datasets. We provide a separate evaluation of the performance of ROCKET on the 'development' datasets and the remaining 'holdout' datasets in "Appendix B".

3.1 Kernels

ROCKET transforms time series using convolutional kernels, as found in typical convolutional neural networks. Essentially all aspects of the kernels are random: length, weights, bias, dilation, and padding. For each kernel, these values are set as follows (as determined by experimentation to produce the highest classification accuracy on the 'development' datasets):

- *Length* Length is selected randomly from $\{7, 9, 11\}$ with equal probability, making kernels considerably shorter than input time series in most cases.
- *Weights* The weights are sampled from a normal distribution, $\forall w \in \boldsymbol{W}$, $w \sim \mathcal{N}(0, 1)$, and are mean centered after being set, $\omega = \boldsymbol{W} - \overline{\boldsymbol{W}}$. As such, most weights are relatively small, but can take on larger magnitudes.
- *Bias* Bias is sampled from a uniform distribution, $b \sim \mathcal{U}(-1, 1)$. Only positive values in the feature maps are relevant for *ppv* (see Sect. 3.2). Bias therefore has the effect that two otherwise similar kernels, but with different biases, can 'highlight' different aspects of the resulting feature maps by shifting the values in a feature map above or below zero by a fixed amount.
- *Dilation* Dilation is sampled on an exponential scale $d = \lfloor 2^x \rfloor, x \sim \mathcal{U}(0, A)$, where $A = log_2 \frac{l_{\text{input}}-1}{l_{\text{kernel}}-1}$, which ensures that the effective length of the kernel, including dilation, is up to the length of the input time series, l_{input}. Dilation allows otherwise similar kernels but with different dilations to match the same or similar patterns at different frequencies and scales.
- *Padding* When each kernel is generated, a decision is made (at random, with equal probability) whether or not padding will be used when applying the kernel. If padding is used, an amount of zero padding is appended to the start and end of each time series when applying the kernel, such that the 'middle' element of the kernel is centered on every point in the time series, i.e., $((l_{\text{kernel}} - 1) \times d)/2$. Without padding, kernels are not centered at the first and last $\lfloor l_{\text{kernel}}/2 \rfloor$ points of the time series, and 'focus' on patterns in the central regions of time series whereas with padding, kernels also match patterns at the start or end of time series (see also Sect. 3.4.1).

Stride is always one. We do not apply a nonlinearity such as the rectified linear unit (or ReLU) to the resulting feature maps. Note that the parameters for the weights and bias have been set based on the assumption that, as is standard practice, input time series have been normalised to have a mean of zero and a standard deviation of one (see generally Dau et al. 2019).

The above kernel parameters, having been determined through experimentation to produce the highest classification accuracy on the 'development' datasets (see

Sect. 4.3), form an intrinsic part of ROCKET and do not need to be 'tuned' for new datasets. In effect, the only hyperparameter for ROCKET is the number of kernels, k. However, in practice, there is no need to change k from its default value of 10,000. In setting k, there is a tradeoff between classification accuracy and computation time. Generally speaking, a larger value of k results in higher classification accuracy (see Sect. 4.3.1), but at the expense of proportionally longer computation: the complexity of the transform is linear with respect to k. However, even with a very large number of kernels, ROCKET is extremely fast.

As demonstrated in the sensitivity analysis in Sect. 4.3, below, there are several alternative configurations which produce similar classification accuracy to the default kernel parameters set out above. Overall, this suggests that our method is likely to generalise well to new problems, and that the kernel parameters are relatively 'uninformative' in the Bayesian sense of the word.

3.2 Transform

Each kernel is applied to each input time series, producing a feature map. The convolution operation involves a sliding dot product between a kernel and an input time series. The result of applying a kernel, ω, with dilation, d, and bias, b, to a given time series, X, from position i in X, is given by (see, e.g., Bai et al. 2018):

$$X_i * \omega = \left(\sum_{j=0}^{l_{\text{kernel}}-1} X_{i+(j \times d)} \times \omega_j \right) + b.$$

ROCKET computes two aggregate features from each feature map, producing two real-valued numbers as features per kernel, and composing our transform:

- the maximum value (broadly speaking, equivalent to global max pooling); and
- the proportion of positive values (or *ppv*), i.e., $ppv(\boldsymbol{Z}) = \frac{1}{n}\sum_{i=0}^{n-1}[z_i > 0]$, where $\boldsymbol{Z}$ is the output of the convolution operation.

Pooling, including global average pooling (Lin et al. 2014), and global max pooling (Oquab et al. 2015), is used in convolutional neural networks for dimensionality reduction and spatial or temporal invariance (Boureau et al. 2010).

The other aggregated feature, *ppv*, directly captures the proportion of the input which matches a given pattern, i.e., for which the output of the convolution operation is positive. The *ppv* works in conjunction with the bias term. The bias term acts as a kind of 'threshold' for *ppv*. A positive bias value means that *ppv* captures the proportion of the input reflecting even 'weak' matches between the input and a given pattern, while a negative bias value means that *ppv* only captures the proportion of the input reflecting 'strong' matches between the input and the given pattern. We found that *ppv* produces meaningfully higher classification accuracy than other features, including the mean (broadly equivalent to global average pooling).

For each kernel, ROCKET produces two features: *ppv* and max. Accordingly, for k kernels, ROCKET produces $2k$ features per time series, and for 10,000 kernels (the default), ROCKET produces 20,000 features. For smaller datasets—in fact, for all of the datasets in the UCR archive—the number of features is therefore possibly much larger than either the number of examples in the dataset or the number of elements in each time series.

Nevertheless, we find that the features produced by ROCKET provide for high classification accuracy when used as the input for a linear classifier, even for datasets where the number of features dwarfs both the number of examples and the length of the time series.

3.3 Classifier

The transformed features are used to train a linear classifier. ROCKET can, in principle, be used with any classifier. We have found that ROCKET is very effective when used in conjunction with linear classifiers, which have the capacity to make use of a small amount of information from each of a large number of features. In our experiments, we use either a ridge regression classifier or logistic regression, depending on dataset size. We suggest using the ridge regression classifier where the number of training examples is less than the number of features (i.e., by default, for less than 20,000 training examples), and otherwise using logistic regression.

Logistic regression ROCKET can be used with logistic regression and stochastic gradient descent. This is particularly suitable for very large datasets because it provides for fast training with a fixed memory cost (fixed by the size of each minibatch). The transform can be performed on each minibatch, or on larger tranches of the dataset which are then divided further into minibatches for training.

Ridge regression However, for all of the datasets in the UCR archive we use a ridge regression classifier, where a ridge regression model is trained for each class in a ‘one versus rest’ fashion, with L_2 regularisation.

Regularisation is critically important where the number of features is significantly greater than the number of training examples, allowing for the optimisation of linear models, and preventing pathological behaviour in iterative optimisation, e.g., for logistic regression (see Goodfellow et al. 2016, pp. 232–233). The ridge regression classifier can exploit generalised cross-validation to determine an appropriate regularisation parameter quickly (see Rifkin and Lippert 2007). We find that, for smaller datasets, a ridge regression classifier is significantly faster in practice than logistic regression, while still achieving high classification accuracy. We find that, for smaller datasets, it is significantly more challenging and time consuming to optimise the multiple hyperparameters for logistic regression (minibatch size, learning rate, regularisation, etc.) to achieve the same classification accuracy as the ridge regression classifier.

3.4 Complexity analysis

The computational complexity of ROCKET has two aspects: (1) the complexity of the transform itself; and (2) the complexity of the linear classifier trained using the transformed features.

3.4.1 Transform

The transform itself is linear in relation to both: (a) the number of examples; and (b) the length of the time series in a given dataset. Formally, the computational complexity of the transform is $O(k \cdot n \cdot l_{\text{input}})$, where k is the number of kernels, n is the number of examples, and l_{input} is the length of the time series. The transform must be applied to both training and test sets.

The convolution operation can be implemented in more than one way, including as a matrix multiplication typical of implementations for convolutional neural networks, and using the fast Fourier transform (see Goodfellow et al. 2016, ch. 9). We implement ROCKET simply, ‘sliding’ each kernel along each time series and computing the dot product at each location. This involves repeated elementwise multiplication and summation, the complexity of which is dictated by the length of the time series, and the length of the kernels (that is, the number of weights in the kernels). The length of the kernels for ROCKET is limited to, at most, 11. Accordingly, kernel length is a constant factor for the purpose of this analysis.

Dilation increases the effective size of a kernel. Accordingly, where no padding is used, dilation reduces computational complexity. Without padding, the convolution is computed with the first element of the kernel starting at the first element of the time series, and ends once the last element of the kernel reaches the last element of the time series. In an extreme case, for the largest values of dilation, the kernel will ‘fill’ the entire time series, and the number of computations will be the number of weights in the kernel. However, padding is applied randomly with equal probability, so the reduction in complexity is a constant factor.

Where padding is used, dilation has no effect on complexity: the same number of computations are required regardless of dilation or the effective size of the kernel. Regardless of dilation, the kernel is centered on the first element of the time series, and ‘slides’ the same number of elements along the time series.

Accordingly, for k kernels and n time series, each of length l_{input}, the complexity of the transform is $O(k \cdot n \cdot l_{\text{input}})$. For datasets with time series of different lengths, this could be taken to represent average complexity for an average length of l_{input}, or worst-case complexity for a maximum length of l_{input}.

3.4.2 Classifier

Logistic regression and stochastic gradient descent The complexity of stochastic gradient descent is proportional to the number of parameters (dictated by the number of features and the number of classes), but is linear in relation to the number of training examples (Bottou et al. 2018). Further, the rate of convergence is not determined by the number of training examples. For large datasets, convergence may occur in a single pass of the data, or even without using all of the training data (Goodfellow et al. 2016, pp. 286–288; Bottou et al. 2018).

Ridge regression In practice, the ridge regression classifier is significantly faster than logistic regression on smaller datasets because it can make use of so-called generalised cross-validation to determine appropriate regularisation. The implementation used here employs eigen decomposition where there are more features than training examples, or singular value decomposition otherwise, with effective complexity of $O(n^2 \cdot f)$ and $O(n \cdot f^2)$ respectively (see Dongarra et al. 2018), where n is the number of training examples and f is the number of features.

This makes the ridge regression classifier less scalable for large datasets. This also requires the complete transform, and does not work incrementally. In practice, these limitations do not affect any of the datasets in the UCR archive. For larger datasets, where the importance of regularisation decreases, and it is appropriate to perform the transform incrementally, the benefit of using the ridge regression classifier wanes, and training with stochastic gradient descent makes more sense.

3.5 Memory

The ridge regression classifier requires the transform for the entire training set. The amount of memory required for the transform depends on training set size, n, and the number of kernels, k. By default, for $k = 10{,}000$ (i.e., 20,000 features), the transform requires $20{,}000 \times 8 \times n$ bytes, or approximately 160 MB per 1000 training examples or, equivalently, approximately 1.6 GB per 10,000 training examples. In practice, for all but one of the 'bake off' datasets, less than 4GB of memory is required for training the ridge regression classifier (including the memory required to store the transform itself). For the 'bake off' dataset with the largest number of training examples, *ElectricDevices*, less than 8 GB is required. In contrast, for logistic regression trained using minibatch gradient descent, the transform can be applied per minibatch, so the amount of memory required is proportional to minibatch size (e.g., approximately 40 MB would required for a minibatch size of 256). For both classifiers, test examples can be transformed as required.

3.6 Limitations

One limitation of ROCKET is that a large number of kernels is required in order to achieve the highest classification accuracy which, in turn, limits the choice of classifier to those classifiers that are effective for a very large number of features (including the ridge regression classifier and logistic regression). An additional limitation is that in using fixed, random kernels, learning is likely to 'saturate' at some point for very large datasets. We would expect more typical convolutional neural network architectures with learned kernels, such as InceptionTime, to achieve higher accuracy on very large datasets, albeit at much greater computational expense. However, ROCKET is at least usable for very large datasets, whereas many existing methods for time series classification are not, and we note that accuracy on the Satellite Image Time Series dataset only seems to plateau after approximately half a million training examples (see Sect. 4.2). ROCKET is currently only configured to work with univariate time series. The extension of ROCKET to multivariate time series and the application of ROCKET to very large datasets is the intended focus of future work.

4 Experiments

We evaluate Rocket on the UCR archive (Sect. 4.1), demonstrating that Rocket is competitive with current state-of-the-art methods, obtaining the best mean rank over the 85 ‘bake off’ datasets.

We evaluate scalability in terms of both training set size and time series length (Sect. 4.2), demonstrating that Rocket is orders of magnitude faster than current methods. We also evaluate the effect of different kernel parameters (Sect. 4.3), showing that several alternative configurations of Rocket perform similarly well, which is a good indication of the power of the idea, rather than of its fine-tuning. Unless otherwise stated, all experiments use 10,000 kernels.

The experiments on the datasets in the UCR archive are performed using Rocket in conjunction with a ridge regression classifier, and the experiment in relation to training set size is performed using Rocket integrated with logistic regression. The experiments on the UCR archive were conducted on a cluster (but using a single CPU core per experiment, not parallelised for speed). The experiments in relation to scalability (both time series length and training set size) were performed locally using an Intel Core i5-5200U dual-core processor.

4.1 UCR archive

4.1.1 Note on the datasets in the UCR archive

The UCR archive is a collection of datasets for time series classification (Dau et al. 2019). A seminal paper, Bagnall et al. (2017), conducted thorough comparative benchmarking of a large number of methods for time series classification on the 85 datasets in the archive as of 2017. We call these the ‘bake off’ datasets after the title of that paper. A further 43 datasets were added to the archive in 2018. We call these the ‘additional 2018 datasets’ (so far there are few published benchmark results for these datasets).

Each dataset in the archive has a predefined training/test split. However, one issue with using only a single training/test split is that there is a possibility that methods ‘tuned’ (inadvertently or otherwise) for this training/test split may not generalise well to other tasks (Bagnall et al. 2017; Dau et al. 2019). To this end, Bagnall et al. (2017) used resamples of the datasets to assess performance. We have performed experiments using both the original training/test splits, as well as ten resamples of the relevant datasets [using the same first ten resamples, excluding the original training/test split, as in Bagnall et al. (2017)].

4.1.2 ‘Bake Off’ datasets

We evaluate Rocket on the 85 ‘bake off’ datasets from the UCR archive on the original training/test split for each dataset. The results presented for Rocket are mean results over 10 runs, using a different set of random kernels for each run.

We compare Rocket to existing state-of-the-art methods for time series classification, namely, BOSS, Shapelet Transform, Proximity Forest, ResNet, and HIVE-COTE. We also compare Rocket with two more recent methods, InceptionTime and TS-CHIEF, that have been demonstrated to be competitive with HIVE-COTE, while being more scalable. The results for BOSS, Shapelet Transform, and HIVE-COTE are taken from Bagnall et al. (2019).

For comparability with other published results, we compare Rocket to the other methods on all 85 'bake off' datasets. However, as noted above, to make sure we didn't overfit the UCR archive, Rocket was developed using a subset of 40 randomly-selected datasets. Separate rankings for the 40 'development' datasets, as well as the remaining 45 'holdout' datasets, are provided in "Appendix B".

Figure 1 on page 1 gives the mean rank for each method included in the comparison. Classifiers for which the difference in pairwise classification accuracy is not statistically significant, as determined by a Wilcoxon signed-rank test with Holm correction (as a post hoc test to the Friedman test), are connected with a black line (see Demšar 2006; García and Herrera 2008; Benavoli et al. 2016). The relative accuracy of Rocket and each of the other methods included in the comparison is shown in Fig. 13, "Appendix A".

Figure 1 on page 1 shows that Rocket is competitive with (in fact, ranks slightly ahead of) HIVE-COTE, TS-CHIEF, and InceptionTime, although the difference in accuracy between Rocket, HIVE-COTE, InceptionTime, and TS-CHIEF is not statistically significant. TS-CHIEF ranks ahead of Rocket on the 45 'holdout' datasets (Fig. 15, "Appendix B"), but the difference is not significant.

Resamples We also evaluate Rocket on ten resamples of the 'bake off' datasets. The results are set out in "Appendix D". The results for the resamples are essentially the same as for the original training/test splits. In fact, while HIVE-COTE ranks ahead of Rocket, Rocket is 'stronger' against both HIVE-COTE and TS-CHIEF in terms of win/draw/loss on the resampled 'bake off' datasets than on the original training/test splits.

Other Methods We also compare Rocket against four recently-proposed scalable methods for time series classification (see Sect. 2.2), namely, MrSEQL, cBOSS, MiSTiCl, and catch22. The results are set out in "Appendix E". The results show that Rocket is significantly more accurate and, with one exception, catch22, more scalable than these methods. Rocket is considerably ahead of the most accurate of these methods, MrSEQL, in terms of win/draw/loss (54/8/23) on the 'bake off' datasets, and Rocket is approximately an order of magnitude faster in terms of total compute time than MrSEQL, cBOSS, and MiSTiCl.

4.1.3 Additional 2018 datasets

We have also evaluated Rocket on the 43 additional datasets in the UCR archive as of 2018. Figure 2 shows the mean rank of Rocket versus InceptionTime, TS-CHIEF, Proximity Forest, and ResNet on the additional 2018 datasets. Figure 2 shows that Rocket is competitive with (in fact, ranks just ahead of) InceptionTime and TS-

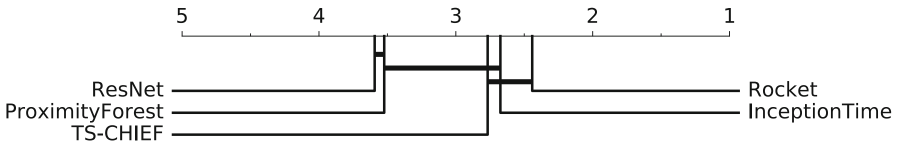


Fig. 2 Mean rank of ROCKET versus state-of-the-art classifiers on the additional 2018 datasets

CHIEF, although the difference in accuracy is not statistically significant. The relative accuracy of ROCKET and each of the other methods is shown in Fig. 14, "Appendix A".

We note that there are currently few published results for state-of-the-art methods on these datasets. For comparability with the results for the other methods, we have used a version of these datasets where missing values have been interpolated and variable-length time series have been padded with 'low amplitude random [noise]' to the same length as the longest time series (Dau et al. 2018, p. 16).

Resamples We also compare ROCKET, Proximity Forest, and TS-CHIEF on ten resamples of the additional 2018 datasets (published results are not available for other methods for these resamples). The results are set out in "Appendix D". Similarly as for the 'bake off' datasets, the results for these methods for the ten resamples are essentially the same as for the original training/test splits.

Other methods As for the 'bake off' datasets, ROCKET is significantly more accurate than MrSEQL, cBOSS, MiSTiCl, or catch22 on the additional 2018 datasets and, with the exception of catch22, considerably faster (see "Appendix E"). Again, ROCKET is substantially ahead of the most accurate of these methods in terms of win/draw/loss (32/0/11). ROCKET is 4 times faster than cBOSS, 16 times faster than MrSEQL, and almost 22 times faster than MiSTiCl on these datasets.

4.2 Scalability

4.2.1 Training set size

Following Lucas et al. (2019), Shifaz et al. (2020), and Ismail Fawaz et al. (2019c), we evaluate scalability in terms of training set size on increasingly larger subsets (up to approximately 1 million time series) of the Satellite Image Time Series dataset (see Petitjean et al. 2012). The time series in this dataset represent a vegetation index, calculated from spectral data acquired by the Formosat-2 satellite, and the classes represent different land cover types. The aim in classifying these time series is to map different vegetation profiles to different types of crops and forested areas. Each time series has a length of 46.

For the purpose of this experiment, we use ROCKET in conjunction with logistic regression (*not* the ridge regression classifier: see Sects. 3.3 and 3.4). The transform is performed in tranches, further divided into minibatches for training. Each time series is normalised to have a zero mean and unit standard deviation.

We shuffle the original training set once, and draw increasingly-large subsets from the shuffled training data. We train the model for at least one epoch for each subset

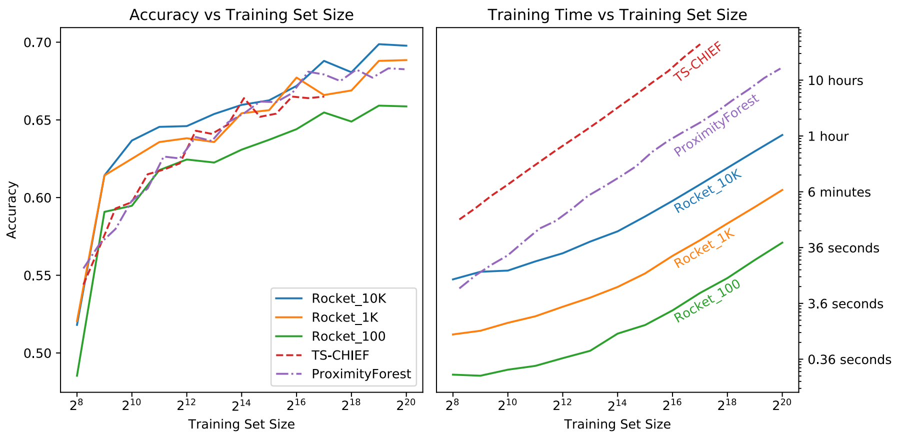


Fig. 3 Accuracy (left) and training time (right) versus training set size for the Satellite Image Time Series dataset

size. To prevent overfitting, we stop training (after the first epoch) if validation loss has failed to improve after 20 updates. In practice, while training may continue for 40 or 50 epochs for smaller subset sizes, training converges within a single pass for anything more than approximately 16,000 training examples. Validation loss is computed on a separate validation set (the same set of 2048 examples for all subset sizes).

Optimisation is performed using Adam (Kingma and Ba 2015). We perform a minimal search on the initial learning rate to ensure that training loss does not diverge. The learning rate is halved if training loss fails to improve after 100 updates (only relevant for larger subset sizes).

We have run ROCKET in three guises: with 100, 1000, and 10,000 kernels (the default). We compare ROCKET against Proximity Forest and TS-CHIEF, which have already been demonstrated to be fundamentally more scalable than HIVE-COTE (Lucas et al. 2019; Shifaz et al. 2020). (Results for larger quantities of data are not yet available for InceptionTime.)

Figure 3 shows classification accuracy and training time versus training set size for ROCKET, Proximity Forest, and TS-CHIEF. As expected, ROCKET scales linearly with respect to both the number of training examples, and the number of kernels (see Sect. 3.4). With 1000 or 10,000 kernels, ROCKET achieves similar classification accuracy to Proximity Forest and TS-CHIEF. With 100 kernels, ROCKET achieves lower classification accuracy, but takes less than a minute to learn from more than 1 million time series. Even with 10,000 kernels, ROCKET is an order of magnitude faster than Proximity Forest. Training time for smaller subset sizes for ROCKET is dominated by the cost of the transform for the validation set, which is why training time is 'flat' for smaller subset sizes.

Other methods We also compare the scalability of ROCKET against MrSEQL, cBOSS, MiSTiCl, and catch22. The results are set out in "Appendix E". MrSEQL, cBOSS, and MiSTiCl are all fundamentally less scalable than ROCKET, and all four methods are less accurate. By approximately 32,000 training examples, MrSEQL is approximately 75 times slower than ROCKET, MiSTiCl is approximately 200 times slower, and cBOSS

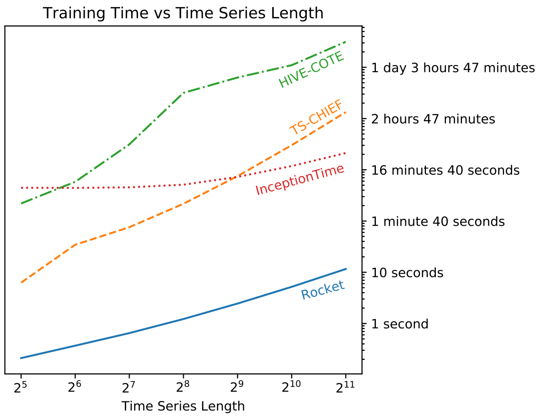


Fig. 4 Training time versus time series length

is more than 300 times slower. While slower than catch22 with its default settings (i.e., 10,000 kernels), restricted to 100 kernels ROCKET is an order of magnitude faster than catch22 and still significantly more accurate.

4.2.2 Time series length

Following Shifaz et al. (2020) and Ismail Fawaz et al. (2019c), we evaluate scalability in terms of time series length using the *InlineSkate* dataset from the UCR archive. We use ROCKET in the same configuration as for the other datasets in the UCR archive (that is, using the ridge regression classifier and 10,000 kernels). Results for HIVE-COTE and TS-CHIEF are taken from Shifaz et al. (2020).

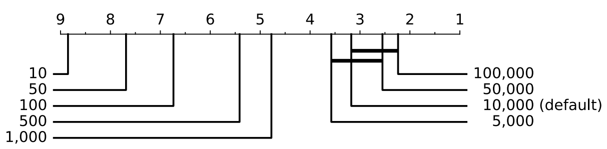
Figure 4 shows training time versus time series length for ROCKET, TS-CHIEF, HIVE-COTE, and InceptionTime. The difference in training time between ROCKET and TS-CHIEF is substantial. ROCKET takes approximately as long to train on time series of length 2048 as TS-CHIEF takes for time series of length 32, and is approximately three orders of magnitude faster for the longest time series length.

ROCKET is considerably faster than InceptionTime as well. However, fundamental scalability is likely to be similar, given that both InceptionTime and ROCKET are based on convolutional architectures.

Other methods The scalability of ROCKET, cBOSS, and catch22 in terms of time series length appears to be similar—approximately linear in time series length (see “Appendix E”). MrSEQL and MiSTiCl are less scalable. MrSEQL, cBOSS, and MiSTiCl are all slower than ROCKET for a given time series length.

4.3 Sensitivity analysis

We explore the effect of different kernel parameters on classification accuracy. We compare the accuracy of the default configuration (i.e., using the parameters specified in Sect. 3) against different choices for the number of kernels, length, weights and bias, dilation, padding, and output features. (As noted above, the default kernel parameters



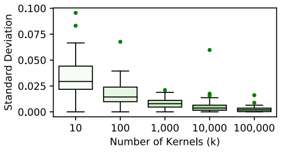


Fig. 5 Mean ranks (left), and variance in accuracy (right), versus *k*

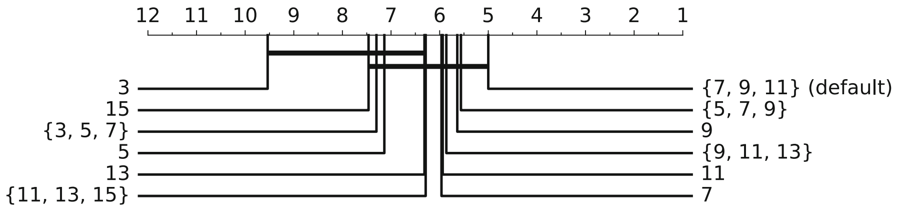


Fig. 6 Mean ranks for different choices in terms of kernel length

form an intrinsic part of ROCKET and do not need be 'tuned' for new datasets.) In each case, only the given parameter (e.g., length) is varied, keeping all other parameters fixed at their default values. The comparison is made on the 'development' datasets. The results are mean results over 10 runs, using a different set of random kernels per run.

In most cases, alternative configurations represent a relatively subtle change from the default configuration. Unsurprisingly, therefore, in many cases one or more alternative choices for the relevant parameter produces similar accuracy to the baseline configuration. In other words, ROCKET is relatively robust to different choices for many parameters. However, it is clear that dilation and *ppv*, in particular, are two key aspects of the performance of the method.

4.3.1 Number of kernels

We evaluate increasing numbers of kernels between 10 and 100,000. Figure 5 shows the effect of the number of kernels, k, on accuracy. Clearly, increasing the number of kernels improves accuracy. However, the actual difference in accuracy between, for example, $k = 5000$ and $k = 10{,}000$, is relatively small. Indeed, $k = 5000$ produces higher accuracy on some datasets (Fig. 17, "Appendix C"). Nevertheless, $k = 10{,}000$ is noticeably ahead in terms of win/draw/loss (24/5/11).

Even though ROCKET is nondeterministic, the variability in accuracy is reasonably low for large numbers of kernels. Unsurprisingly, standard deviation in accuracy diminishes as k increases. The median standard deviation across the 40 'development' datasets is 0.0037 for $k = 10{,}000$, and 0.0019 for $k = 100{,}000$.

4.3.2 Kernel length

We vary kernel length, comparing the baseline (selecting length randomly from $\{7, 9, 11\}$) to:

- fixed lengths of 3, 5, 7, 9, 11, 13, and 15; and
- selecting length randomly from $\{3, 5, 7\}$, $\{5, 7, 9\}$, $\{9, 11, 13\}$, and $\{11, 13, 15\}$.

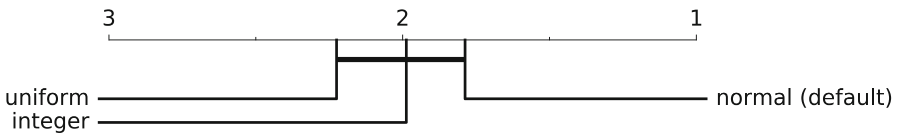


Fig. 7 Mean ranks for different choices in terms of the sampling distribution for the weights

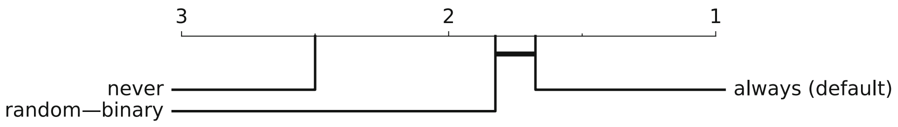


Fig. 8 Mean ranks for different choices in terms of centering

Figure 6 shows the effect of these choices on accuracy. Fixed lengths of 7, 9, and 11, as well as selecting length randomly from $\{5, 7, 9\}$ and $\{9, 11, 13\}$ result in similar accuracy to the default configuration, and the differences are not statistically significant (see also Fig. 18, "Appendix C"). Shorter kernels are undesirable, as they are more likely to be highly correlated (bearing in mind that kernels of the same length still have substantial variety in terms of, e.g., dilation), while longer random kernels are less likely to 'match' patterns in the input, as longer kernels will essentially just represent random noise.

4.3.3 Weights (including centering) and bias

Weights We vary the distribution from which the weights are sampled, comparing the baseline (sampling from a normal distribution) to:

- sampling from a uniform distribution, $\forall w \in \boldsymbol{W}, w \sim \mathcal{U}(-1, 1)$; and
- sampling integer weights uniformly from $\{-1, 0, 1\}$.

Figure 7 shows the effect of these choices on accuracy. While sampling from a normal distribution produces higher accuracy, the actual difference in accuracy is small and not statistically significant (see also Fig. 19, "Appendix C"). While it may seem surprising that weights sampled from only three integer values are so effective, note that kernels are still mean centered by default and have random bias, and there is still substantial variety in terms of length and dilation.
Centering We vary centering, comparing the baseline (always centering) against:

- never centering the kernel weights; and
- centering or not centering at random with equal probability.

Figure 8 shows the effect of these choices on accuracy. It is clear that centering produces higher accuracy, but the difference between always centering and centering at random is very small and not statistically significant. Always centering, however, is noticeably more accurate on some datasets (Fig. 20, "Appendix C").
Bias We vary bias, comparing the baseline (using a uniform distribution) against:

- using zero bias; and
- sampling bias from a normal distribution, $b \sim \mathcal{N}(0, 1)$.

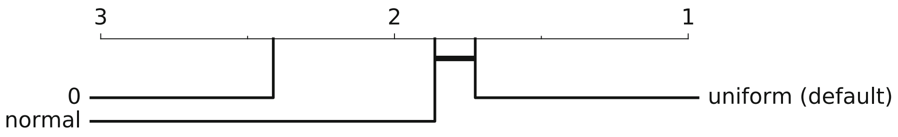


Fig. 9 Mean ranks for different choices in terms of bias

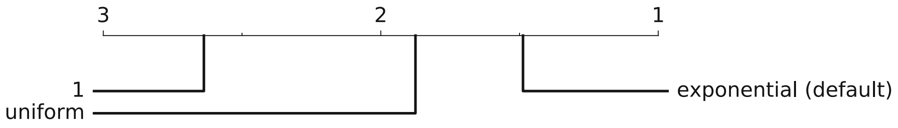


Fig. 10 Mean ranks for different choices in terms of dilation

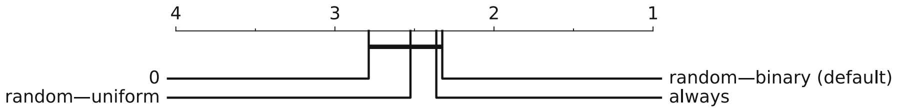


Fig. 11 Mean ranks for different choices in terms of padding

Figure 9 shows the effect of these choices on accuracy. Using a bias term produces higher accuracy, but the difference between sampling bias from a uniform distribution or a normal distribution is relatively small and not statistically significant (see also Fig. 21, "Appendix C".)

4.3.4 Dilation

We vary dilation, comparing the baseline (sampling dilation on an exponential scale) against:

- no dilation (i.e., a fixed dilation of one); and
- sampling dilation uniformly, $d = \lfloor x \rfloor$, $x \sim \mathcal{U}(1, \frac{l_{\text{input}}-1}{l_{\text{kernel}}-1})$.

Figure 10 shows the effect of these choices in terms of accuracy. It is clear that dilation is key to performance. Dilation produces obviously higher accuracy than no dilation. Exponential dilation produces higher accuracy than uniform dilation on most datasets (significantly higher on some datasets), and the difference is statistically significant (see also Fig. 22, "Appendix C").

4.3.5 Padding

We vary padding, comparing the baseline (applying padding at random) against:

- always padding, such that the 'middle' element of a given kernel is centered on the first element of the time series, $p = ((l_{\text{kernel}} - 1) \times d)/2$;
- sampling padding uniformly, $p \sim \mathcal{U}(0, ((l_{\text{kernel}} - 1) \times d)/2)$; and
- never padding.

Figure 11 shows the effect of these choices on accuracy. Padding is superior to not padding, but none of the differences are statistically significant. Different choices produce very similar results (Fig. 23, "Appendix C").

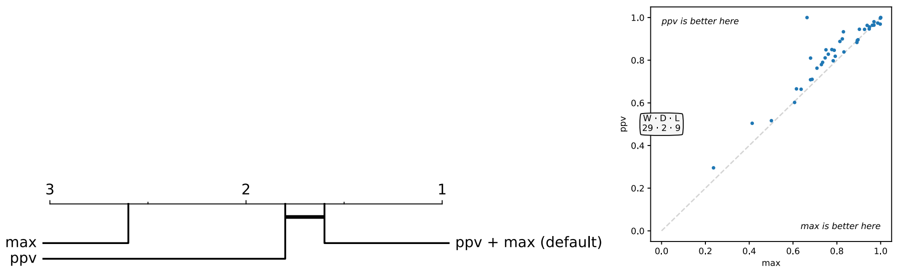


Fig. 12 Mean ranks (left), and relative accuracy (right), *ppv* and max. (‘W’, ‘D’, and ‘L’ signify ‘win’, ‘draw’, and ‘loss’.)

4.3.6 Features

We evaluate the effect of using different output features, i.e., different features produced by the convolution operation between each time series and each convolutional kernel (see Sect. 3.2). We compare the baseline, using both *ppv* and max, against using only *ppv*, and only max. Figure 12 shows the effect of these choices on accuracy. Note that ‘W’, ‘D’, and ‘L’ in Fig. 12 signify ‘win’, ‘draw’, and ‘loss’. It is clear that using only *ppv* is superior to using only max: *ppv* produces substantially higher classification accuracy for the majority of the ‘development’ datasets. In fact, the use of *ppv* has the single biggest effect on accuracy of all the parameters. The combination of *ppv* and max is better again, although the difference between *ppv* and *ppv* plus max is small and not statistically significant (see also Fig. 24, “Appendix C”).

5 Conclusion

Convolutional kernels are a single, powerful instrument which can capture many of the features used by existing methods for time series classification. We show that, rather than learning kernel weights, a large number of random kernels—while in isolation only approximating relevant patterns—in combination are extremely effective for capturing discriminative patterns in time series.

Further, random kernels have very low computational requirements, making learning and classification extremely fast. Our proposed method utilising random convolutional kernels for the purposes of transforming and classifying time series, Rocket, achieves state-of-the-art accuracy with a fraction of the computational expense of existing state-of-the-art methods, and can scale to millions of time series. We also show that Rocket is significantly more accurate and, with one exception, fundamentally more scalable than several recently-proposed scalable methods for time series classification.

Rocket makes key use of the proportion of positive values (or *ppv*) to summarise the output of feature maps, allowing a classifier to weight the prevalence of a pattern in a given time series. To our knowledge, *ppv* has not been used in this way before. We find that this is substantially more effective than a simple maximum as applied in a conventional max pooling operation. It is credible that *ppv* would also be effective for other data types such as images.

In future work, we propose to explore feature selection for Rocket, the application of Rocket to multivariate time series, the application of Rocket beyond time series data, and the use of aspects of Rocket with learned kernels.

Acknowledgements This material is based upon work supported by an Australian Government Research Training Program Scholarship; the Air Force Office of Scientific Research, Asian Office of Aerospace Research and Development (AOARD) under award number FA2386-18-1-4030; and the Australian Research Council under awards DE170100037 and DP190100017. The authors would like to thank Professor Eamonn Keogh and all the people who have contributed to the UCR time series classification archive. Figures showing the ranking of different classifiers and variants of Rocket were produced using code from Ismail Fawaz et al. (2019a).

Appendices

A Relative accuracy

A.1 'Bake Off' datasets

See Fig. 13.

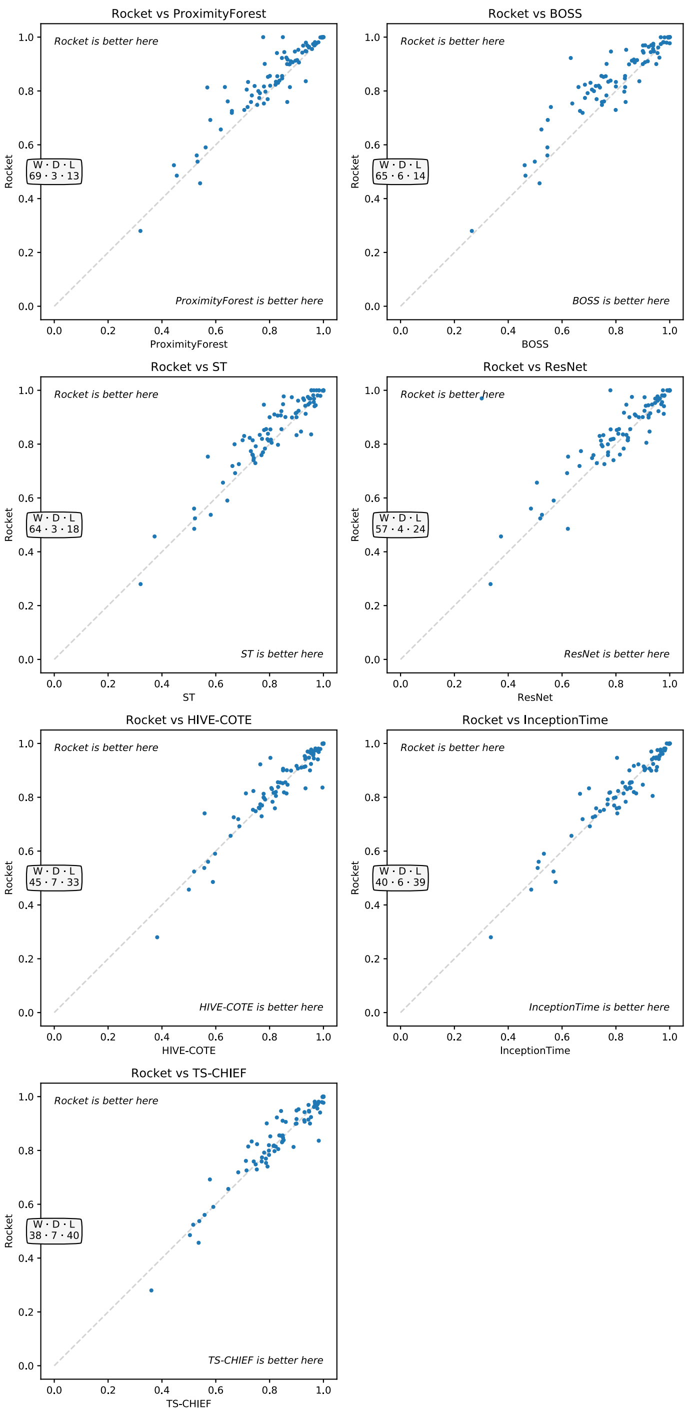


Fig. 13 Relative accuracy of ROCKET versus state-of-the-art classifiers on the 'bake off' datasets

A.2 Additional 2018 datasets

See Fig. 14.

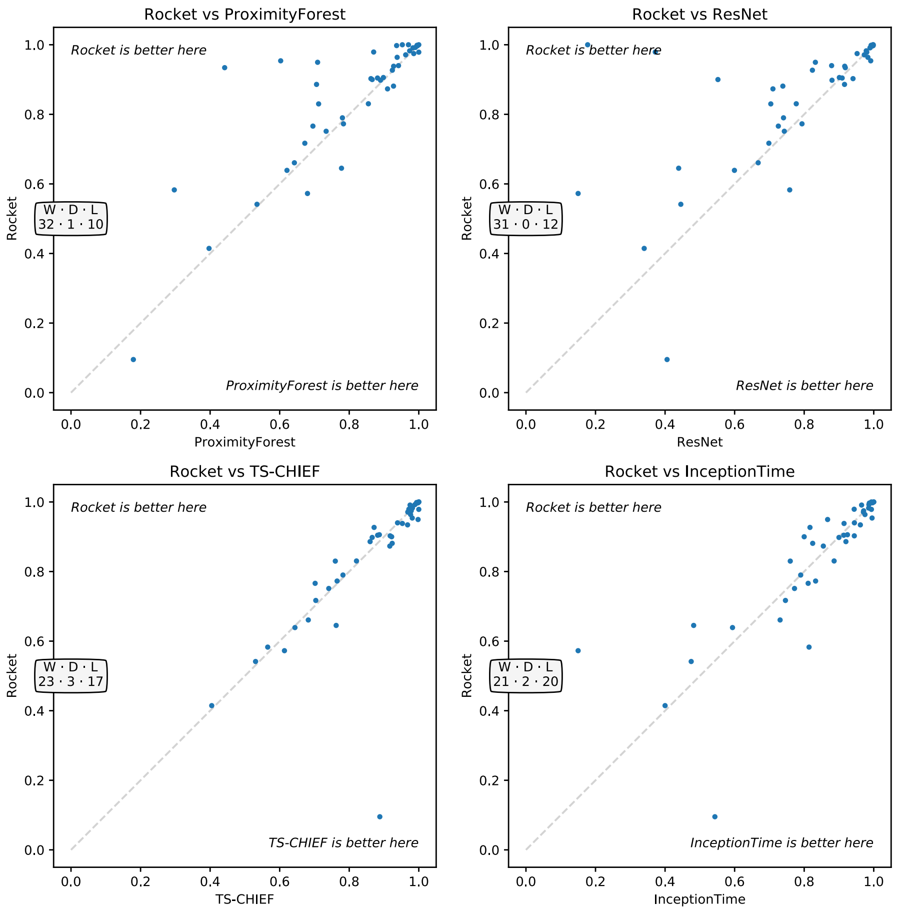


Fig. 14 Relative accuracy of ROCKET versus state-of-the-art classifiers, additional 2018 datasets

B ‘Development’ and ‘Holdout’ datasets

See Figs. 15 and 16.

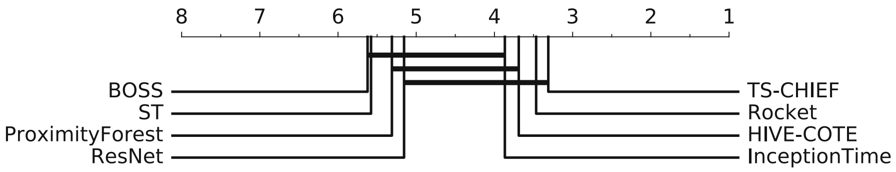


Fig. 15 Mean rank of ROCKET versus state-of-the-art classifiers on the ‘holdout’ datasets

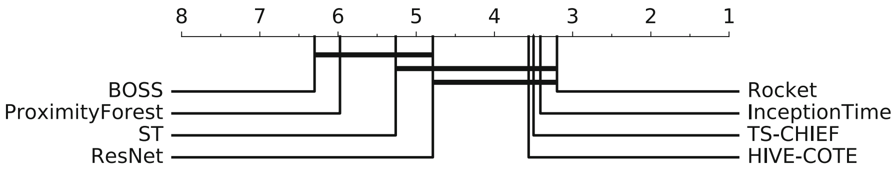
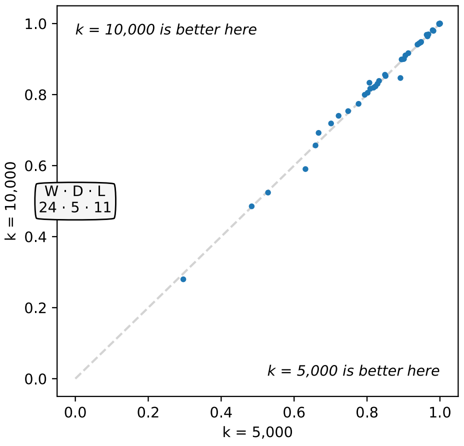


Fig. 16 Mean rank of ROCKET versus state-of-the-art classifiers on the ‘development’ datasets

C Additional plots for the sensitivity analysis

See Figs. 17, 18, 19, 20, 21, 22, 23 and 24.

Fig. 17 Relative accuracy of $k = 10{,}000$ versus $k = 5000$ on the ‘development’ datasets



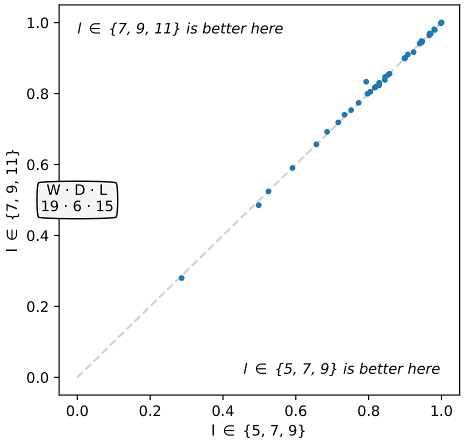


Fig. 18 Relative accuracy of $l \in \{7, 9, 11\}$ versus $l \in \{5, 7, 9\}$ on the 'development' datasets

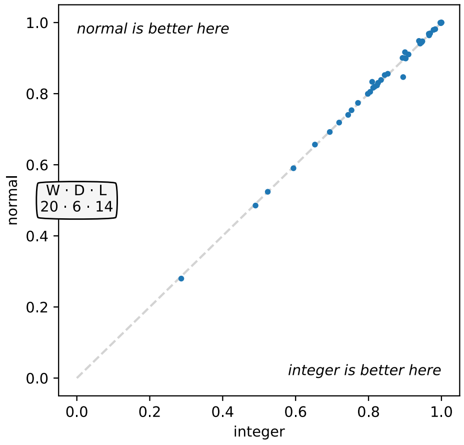


Fig. 19 Relative accuracy, normally-distributed versus integer weights, 'development' datasets

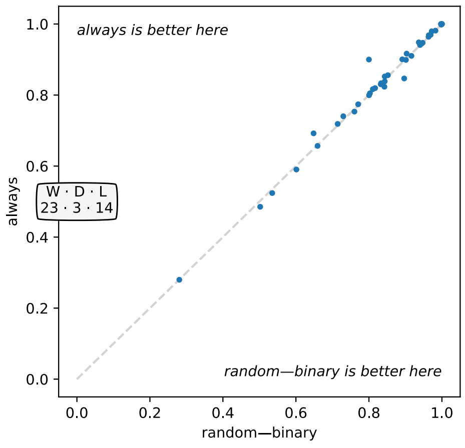


Fig. 20 Relative accuracy of always versus random centering on the 'development' datasets

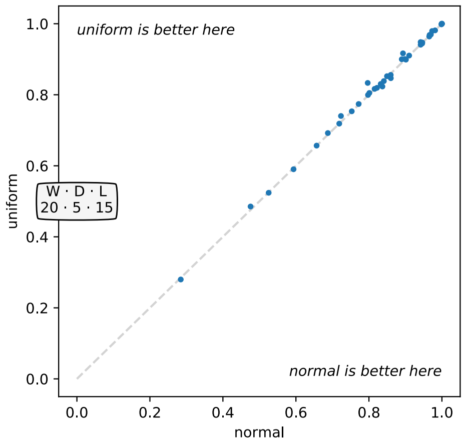


Fig. 21 Relative accuracy, uniformly versus normally-distributed bias, 'development' datasets

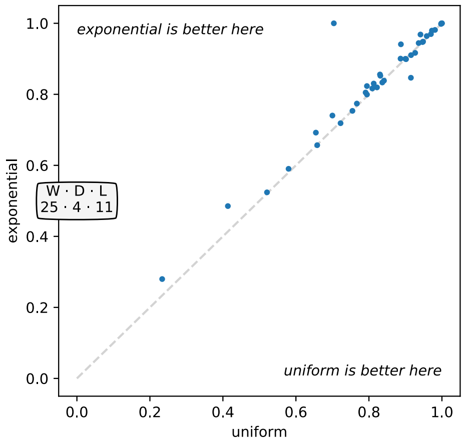


Fig. 22 Relative accuracy of exponential versus uniform dilation on the 'development' datasets

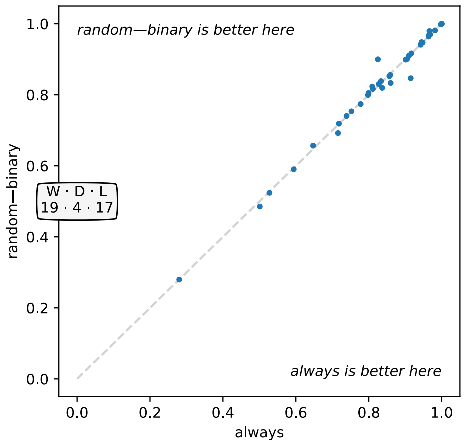


Fig. 23 Relative accuracy of random versus always padding on the ‘development’ datasets

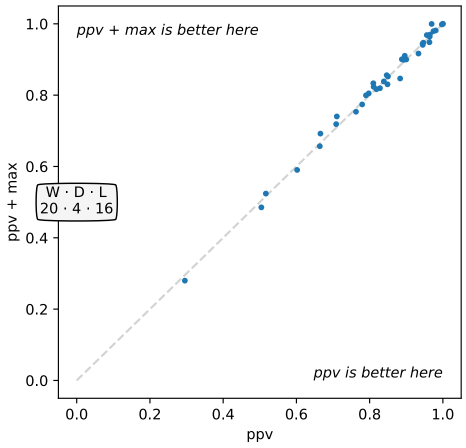


Fig. 24 Relative accuracy of *ppv* and max versus only *ppv* on the ‘development’ datasets

D Resamples

We also evaluate ROCKET on 10 resamples of both the ‘bake off’ and additional 2018 datasets, using the same first 10 resamples (*not* including the original training/test split) as in Bagnall et al. (2017). Figure 25 shows the mean rank of ROCKET versus HIVE-COTE, TS-CHIEF, Shapelet Transform, Proximity Forest and BOSS on the resampled ‘bake off’ datasets. Figure 27 shows the relative accuracy of ROCKET and each of the other methods on the resampled ‘bake off’ datasets. The results for HIVE-COTE, Shapelet Transform, and BOSS are taken from Bagnall et al. (2019). Figures 26 and 28 show the mean rank and relative accuracy of ROCKET versus Proximity Forest and TS-CHIEF for 10 resamples of the additional 2018 datasets (published results are not available for other methods for these resamples).

The results for the resamples and the original training/test splits are very similar for both the ‘bake off’ and additional 2018 datasets. In fact, while HIVE-COTE ranks ahead of ROCKET, ROCKET appears to be ‘stronger’ against both HIVE-COTE and TS-

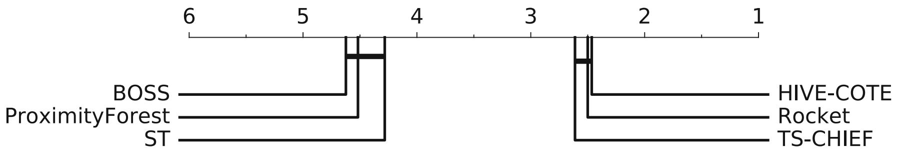


Fig. 25 Mean rank of ROCKET versus other classifiers on the resampled ‘bake off’ datasets

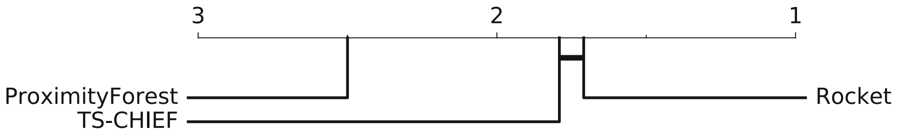


Fig. 26 Mean rank of ROCKET versus other classifiers, resampled additional 2018 datasets

CHIEF on the resamples of the ‘bake off’ datasets than on the original training/test splits. For the resampled ‘bake off’ datasets, ROCKET is ahead of HIVE-COTE in terms of win/draw/loss (47/2/36), as it is for the original training/test split (45/7/33). These results confirm that the results for the original training/test split are sound, and representative of the expected performance of ROCKET relative to the other methods included in the comparison.

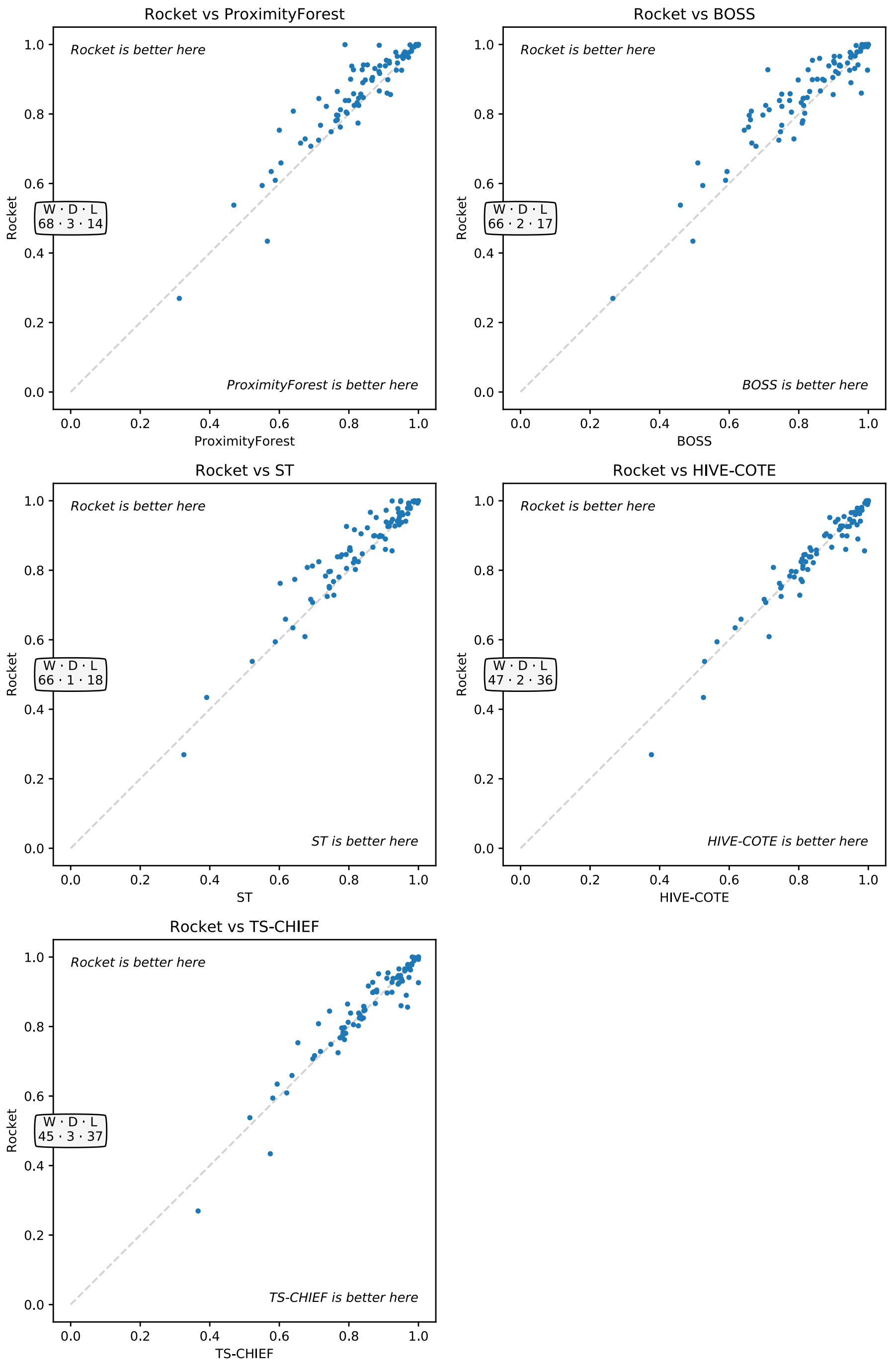


Fig. 27 Relative accuracy of ROCKET versus other classifiers on the resampled 'bake off' datasets

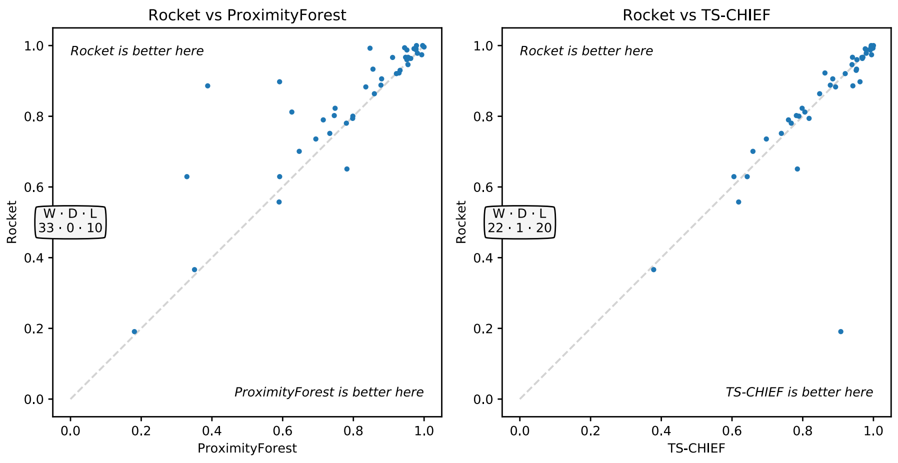


Fig. 28 Relative accuracy of ROCKET versus other classifiers, resampled additional 2018 datasets

E Other methods

We also compare ROCKET against four recently-proposed scalable methods for time series classification (see Sect. 2.2), namely, MrSEQL, cBOSS, MiSTiCl, and catch22. We have run each of these four methods on the 'bake off' datasets, the additional 2018 datasets, and the scalability experiments in terms of both training set size and time series length. We have run each method with its recommended settings per the relevant papers, and using the same experimental conditions as for ROCKET in each case. We have used the Python wrapper for MrSEQL (https://github.com/alan-turing-institute/sktime), the Java version of cBOSS (https://github.com/uea-machine-learning/tsml), the Java version of MiSTiCl (https://github.com/atifraza/MiSTiCl), and the Python wrapper for catch22 (https://github.com/chlubba/catch22).

E.1 'Bake Off' and additional 2018 datasets

Figures 29 and 32 show the mean rank and relative accuracy of ROCKET versus MrSEQL, cBOSS, MiSTiCl, and catch 22 for the 85 'bake off' datasets. Figures 30 and 33 show the same for the additional 2018 datasets. Figure 31 shows total compute time.

The results show that ROCKET is significantly more accurate and, with one exception, more scalable than these methods. ROCKET is considerably ahead of the most accurate of these methods, MrSEQL, in terms of win/draw/loss (54/8/23) on the 'bake off' datasets, and ROCKET is approximately an order of magnitude faster in terms of total compute time than MrSEQL, cBOSS, and MiSTiCl. While catch22 is very fast, it is the least accurate method.

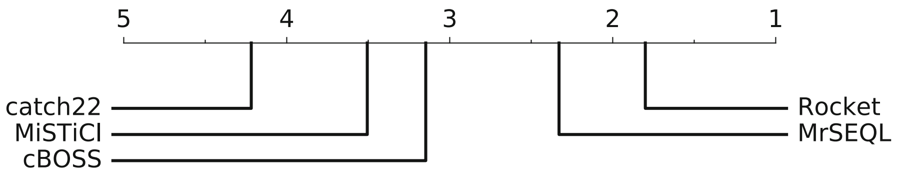


Fig. 29 Mean rank of ROCKET versus other classifiers on the 'bake off' datasets

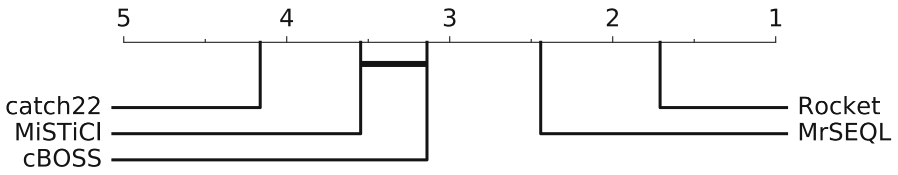


Fig. 30 Mean rank of ROCKET versus other classifiers, additional 2018 datasets

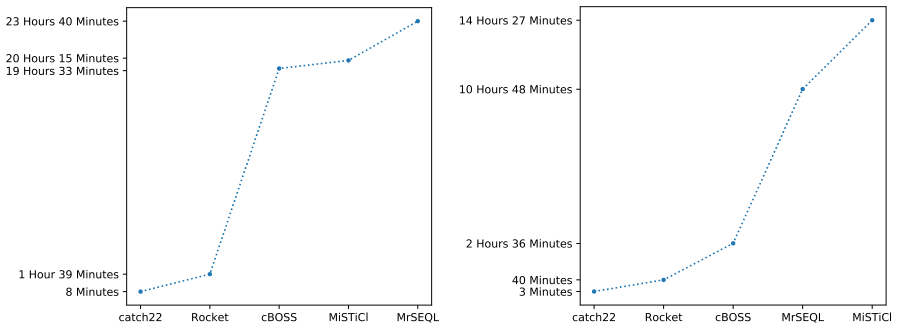


Fig. 31 Total compute time for 'bake off' datasets (left) and additional 2018 datasets (right)

As for the 'bake off' datasets, ROCKET is significantly more accurate than MrSEQL, cBOSS, MiSTiCl, or catch22 on the additional 2018 datasets and, with the exception of catch22, considerably faster. Again, ROCKET is substantially ahead of the most accurate of these methods in terms of win/draw/loss (32/0/11). ROCKET is 4 times faster than cBOSS, 16 times faster than MrSEQL, and almost 22 times faster than MiSTiCl on these datasets. Again, catch22 is the fastest but least accurate method.

Note that while catch22 was originally used in conjunction with a single decision tree, we found that this produced very low accuracy and, of several 'off the shelf' classifiers, random forest produced the highest accuracy. Accordingly, we have used catch22 in conjunction with a random forest classifier. MiSTiCl would not run on the *ElectricDevices* dataset in its published configuration. For this dataset we used MiSTiCl in conjunction with AdaBoost, rather than the default extremely randomised trees. MiSTiCl would not run at all on the *Chinatown* dataset, so it has been ranked behind the other methods for this dataset, and this dataset has been removed from the relevant plot in Fig. 33.

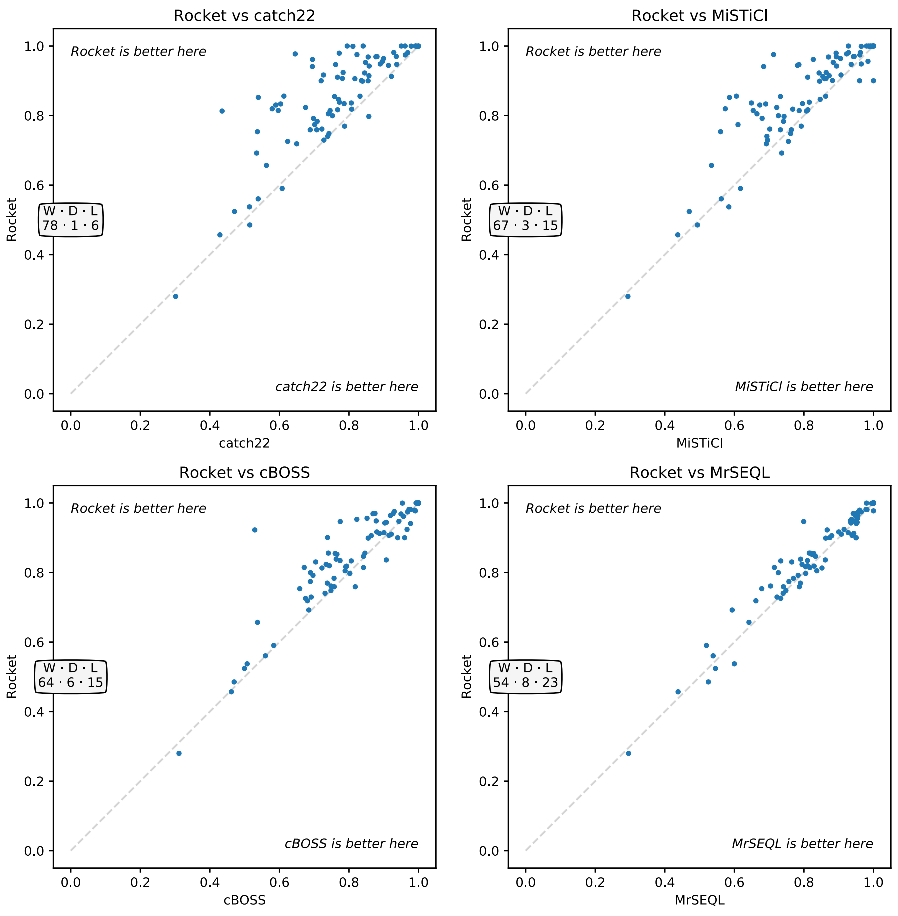


Fig. 32 Relative accuracy of ROCKET versus other classifiers on the 'bake off' datasets

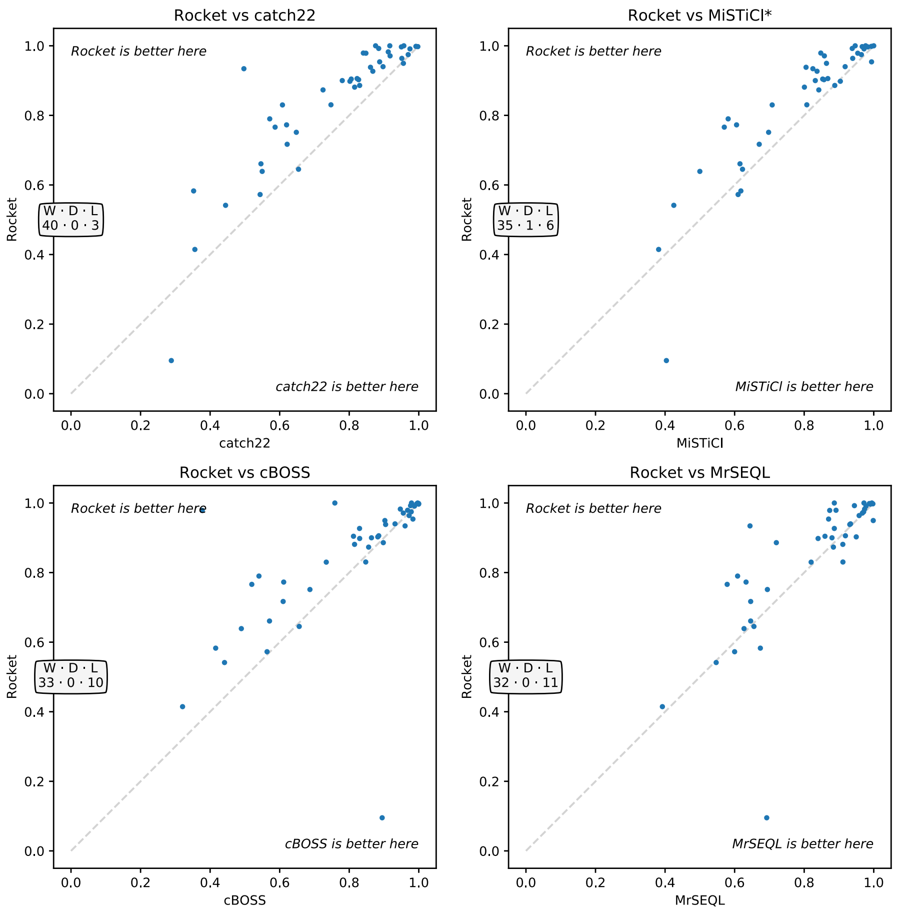


Fig. 33 Relative accuracy of ROCKET versus other classifiers, additional 2018 datasets

E.2 Scalability

E.2.1 Training set size

Figure 34 shows accuracy and training time versus training set size for ROCKET—with 10,000 (default), 1000 and 100 kernels—and the other four methods for the Satellite Image Time Series Dataset. Figure 34 shows that MrSEQL, cBOSS, and MiSTiCl are all fundamentally less scalable than ROCKET in terms of training set size. By approximately 32,000 training examples, MrSEQL is approximately 75 times slower than ROCKET, MiSTiCl is approximately 200 times slower than ROCKET, and cBOSS is more than 300 times slower than ROCKET. Additionally, all four methods are noticeably less accurate than ROCKET for the same training set size. We note that there appears to be a problem with MrSEQL with more than approximately 8000 training examples. While slower than catch22 with its default settings (i.e., 10,000 kernels), in contexts

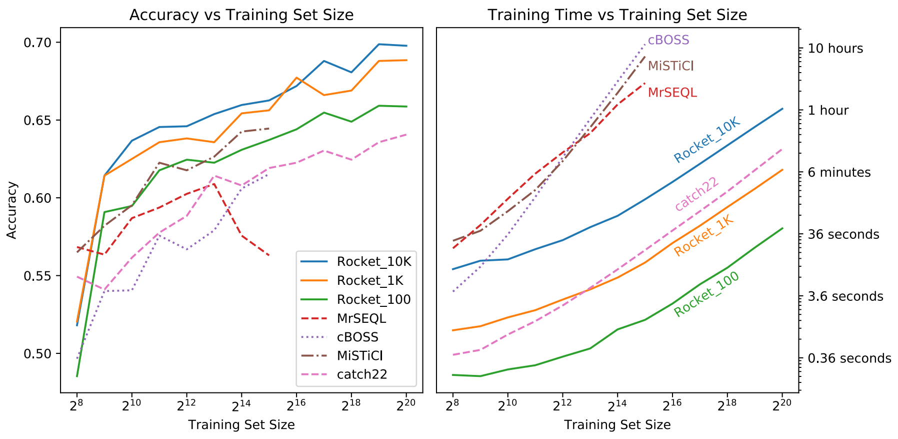


Fig. 34 Accuracy (left) and training time (right) versus training set size

where this speed difference is important, restricted to 100 kernels ROCKET is an order of magnitude faster than catch22 and still significantly more accurate (see Fig. 34).

E.2.2 Time series length

Figure 35 shows training time versus time series length for ROCKET versus the other four methods for the *InlineSkate* dataset. Figure 35 shows that, in practice, the scalability of ROCKET, cBOSS, and catch22 in terms of time series length appears to be similar—that is, approximately linear in time series length. Both MrSEQL and MiSTiCl are less scalable. MrSEQL, cBOSS, and MiSTiCl are all slower than ROCKET for a given time series length.

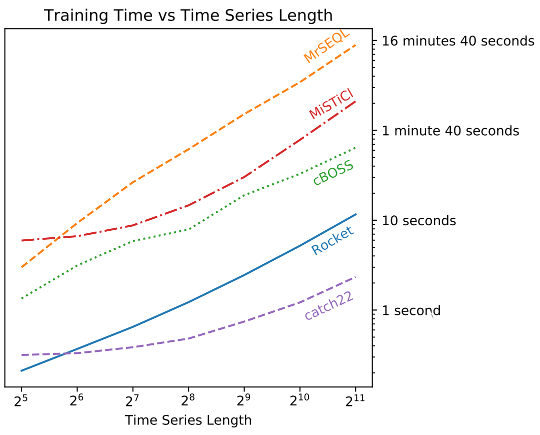


Fig. 35 Training time versus time series length

F Results for 'Bake Off' datasets

See Table 1.

Table 1 Accuracy—'Bake Off' datasets

	Rocket	PF	BOSS	ST	ResNet	HCTE	ITime	CHIEF
Adiac	0.7834	0.7340	0.7647	0.7826	0.8289	0.8107	0.8363	0.7980
ArrowHead	0.8143	0.8754	0.8343	0.7371	0.8446	0.8629	0.8286	0.8229
*Beef	0.8333	0.7200	0.8000	0.9000	0.7533	0.9333	0.7000	0.7333
BeetleFly	0.9000	0.8750	0.9000	0.9000	0.8500	0.9500	0.8500	0.9500
*BirdChicken	0.9000	0.8650	0.9500	0.8000	0.8850	0.8500	0.9500	0.9000
CBF	1.0000	0.9933	0.9978	0.9744	0.9950	0.9989	0.9989	0.9978
*Car	0.8467	0.8467	0.8333	0.9167	0.9250	0.8667	0.9000	0.8500
ChlCon	0.8145	0.6339	0.6609	0.6997	0.8436	0.7120	0.8753	0.7206
CinCECGTorso	0.8362	0.9343	0.8870	0.9543	0.8261	0.9964	0.8514	0.9826
Coffee	1.0000	1.0000	1.0000	0.9643	1.0000	1.0000	1.0000	1.0000
Computers	0.7612	0.6444	0.7560	0.7360	0.8148	0.7600	0.8120	0.7120
*CricketX	0.8195	0.8021	0.7359	0.7718	0.7913	0.8231	0.8667	0.7974
*CricketY	0.8523	0.7938	0.7538	0.7795	0.8033	0.8487	0.8513	0.8026
*CricketZ	0.8559	0.8010	0.7462	0.7872	0.8115	0.8308	0.8590	0.8359
DiaSizRed	0.9699	0.9657	0.9314	0.9248	0.3013	0.9412	0.9314	0.9771
DisPhaOutAgeGro	0.7590	0.7309	0.7482	0.7698	0.7165	0.7626	0.7266	0.7410
DisPhaOutCor	0.7696	0.7928	0.7283	0.7754	0.7710	0.7717	0.7935	0.7862
*DisPhaTW	0.7187	0.6597	0.6763	0.6619	0.6647	0.6835	0.6763	0.6835
ECG200	0.9060	0.9090	0.8700	0.8300	0.8740	0.8500	0.9100	0.8600
*ECG5000	0.9472	0.9365	0.9413	0.9438	0.9342	0.9462	0.9409	0.9458
ECGFiveDays	1.0000	0.8492	1.0000	0.9837	0.9748	1.0000	1.0000	1.0000
Earthquakes	0.7482	0.7540	0.7482	0.7410	0.7115	0.7482	0.7410	0.7482
ElectricDevices	0.7294	0.7060	0.7992	0.7470	0.7291	0.7703	0.7227	0.7524
FaceAll	0.9465	0.8938	0.7817	0.7787	0.8388	0.8030	0.8041	0.8426
FaceFour	0.9773	0.9739	1.0000	0.8523	0.9545	0.9545	0.9659	1.0000
FacesUCR	0.9614	0.9459	0.9571	0.9059	0.9547	0.9629	0.9732	0.9649
*FiftyWords	0.8303	0.8314	0.7055	0.7055	0.7396	0.8088	0.8418	0.8462
*Fish	0.9794	0.9349	0.9886	0.9886	0.9794	0.9886	0.9829	0.9943
*FordA	0.9444	0.8546	0.9295	0.9712	0.9205	0.9644	0.9483	0.9470
*FordB	0.8051	0.7149	0.7111	0.8074	0.9131	0.8235	0.9365	0.8321
GunPoint	1.0000	0.9973	1.0000	1.0000	0.9907	1.0000	1.0000	1.0000
Ham	0.7257	0.6600	0.6667	0.6857	0.7571	0.6667	0.7143	0.7143
HandOutlines	0.9424	0.9214	0.9027	0.9324	0.9111	0.9324	0.9595	0.9297
*Haptics	0.5240	0.4445	0.4610	0.5227	0.5188	0.5195	0.5682	0.5162
*Herring	0.6922	0.5797	0.5469	0.6719	0.6188	0.6875	0.7031	0.5781

Table 1 continued

	Rocket	PF	BOSS	ST	ResNet	HCTE	ITime	CHIEF
InlineSkate	0.4569	0.5418	0.5164	0.3727	0.3731	0.5000	0.4855	0.5364
*InsWinSou	0.6568	0.6187	0.5232	0.6268	0.5065	0.6551	0.6348	0.6465
*ItaPowDem	0.9696	0.9671	0.9086	0.9475	0.9630	0.9631	0.9679	0.9718
*LarKitApp	0.9005	0.7819	0.7653	0.8587	0.8997	0.8640	0.9067	0.7893
Lightning2	0.7590	0.8656	0.8361	0.7377	0.7705	0.8197	0.8033	0.7705
*Lightning7	0.8233	0.8219	0.6849	0.7260	0.8452	0.7397	0.8082	0.7534
Mallat	0.9559	0.9576	0.9382	0.9642	0.9716	0.9620	0.9629	0.9774
*Meat	0.9483	0.9333	0.9000	0.8500	0.9683	0.9333	0.9500	0.9000
*MedicalImages	0.7995	0.7582	0.7184	0.6697	0.7703	0.7776	0.7987	0.7974
*MidPhaOutAgeGro	0.5903	0.5623	0.5455	0.6429	0.5688	0.5974	0.5325	0.5909
*MidPhaOutCor	0.8385	0.8364	0.7801	0.7938	0.8089	0.8316	0.8351	0.8522
MiddlePhalanxTW	0.5604	0.5292	0.5455	0.5195	0.4844	0.5714	0.5130	0.5584
MoteStrain	0.9146	0.9024	0.8786	0.8970	0.9276	0.9329	0.9034	0.9441
NonInvFetECGTho1	0.9530	0.9066	0.8382	0.9496	0.9454	0.9303	0.9623	0.9074
NonInvFetECGTho2	0.9691	0.9399	0.9008	0.9511	0.9461	0.9445	0.9674	0.9445
*OSULeaf	0.9409	0.8273	0.9545	0.9669	0.9785	0.9793	0.9339	0.9876
*OliveOil	0.9167	0.8667	0.8667	0.9000	0.8300	0.9000	0.8667	0.9000
PhaOutCor	0.8343	0.8235	0.7716	0.7634	0.8390	0.8065	0.8543	0.8485
*Phoneme	0.2799	0.3201	0.2648	0.3207	0.3343	0.3824	0.3354	0.3608
*Plane	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
ProPhaOutAgeGro	0.8556	0.8463	0.8341	0.8439	0.8532	0.8585	0.8537	0.8488
*ProPhaOutCor	0.8990	0.8732	0.8488	0.8832	0.9213	0.8797	0.9313	0.8969
*ProPhaTW	0.8166	0.7790	0.8000	0.8049	0.7805	0.8146	0.7756	0.8146
RefDev	0.5373	0.5323	0.4987	0.5813	0.5253	0.5573	0.5093	0.5387
*ScreenType	0.4853	0.4552	0.4640	0.5200	0.6216	0.5893	0.5760	0.5040
*ShapeletSim	1.0000	0.7761	1.0000	0.9556	0.7794	1.0000	0.9889	1.0000
ShapesAll	0.9068	0.8858	0.9083	0.8417	0.9213	0.9050	0.9250	0.9300
SmaKitApp	0.8184	0.7443	0.7253	0.7920	0.7861	0.8533	0.7787	0.8160
SonAIBORobSur1	0.9225	0.8458	0.6323	0.8436	0.9581	0.7654	0.8835	0.8270
SonAIBORobSur2	0.9126	0.8963	0.8594	0.9339	0.9778	0.9276	0.9528	0.9286
StarLightCurves	0.9810	0.9813	0.9778	0.9785	0.9718	0.9815	0.9792	0.9820
*Strawberry	0.9814	0.9684	0.9757	0.9622	0.9805	0.9703	0.9838	0.9676
*SwedishLeaf	0.9640	0.9466	0.9216	0.9280	0.9563	0.9536	0.9712	0.9664
Symbols	0.9743	0.9616	0.9668	0.8824	0.9064	0.9739	0.9819	0.9799
*SynCon	0.9997	0.9953	0.9667	0.9833	0.9983	0.9967	0.9967	1.0000
*ToeSeg1	0.9684	0.9246	0.9386	0.9649	0.9627	0.9825	0.9693	0.9693
ToeSeg2	0.9238	0.8623	0.9615	0.9077	0.9062	0.9538	0.9385	0.9538

Table 1 continued

	Rocket	PF	BOSS	ST	ResNet	HCTE	ITime	CHIEF
*Trace	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
TwoLeadECG	0.9991	0.9886	0.9807	0.9974	1.0000	0.9965	0.9956	0.9965
TwoPatterns	1.0000	0.9996	0.9930	0.9550	0.9999	1.0000	1.0000	1.0000
UWavGesLibAll	0.9754	0.9723	0.9389	0.9422	0.8595	0.9685	0.9545	0.9687
UWavGesLibX	0.8547	0.8286	0.7621	0.8029	0.7805	0.8398	0.8247	0.8417
*UWavGesLibY	0.7740	0.7615	0.6851	0.7303	0.6701	0.7655	0.7688	0.7716
UWavGesLibZ	0.7919	0.7640	0.6949	0.7485	0.7501	0.7831	0.7697	0.7797
*Wafer	0.9982	0.9955	0.9948	1.0000	0.9986	0.9994	0.9987	0.9989
Wine	0.8130	0.5685	0.7407	0.7963	0.7444	0.7778	0.6667	0.8889
*WordSynonyms	0.7534	0.7787	0.6379	0.5705	0.6224	0.7382	0.7555	0.7868
*Worms	0.7403	0.7182	0.5584	0.7403	0.7909	0.5584	0.8052	0.7922
WormsTwoClass	0.7974	0.7844	0.8312	0.8312	0.7468	0.7792	0.7922	0.8182
*Yoga	0.9104	0.8786	0.9183	0.8177	0.8702	0.9177	0.9057	0.8483

The ‘development’ datasets are marked with an asterisk

G Results for additional 2018 datasets

See Table 2.

Table 2 Accuracy—additional 2018 datasets

	Rocket	PF	ResNet	CHIEF	ITime
ACSF1	0.8860	0.7060	0.9160	0.8600	0.9200
AllGesWiiX	0.7900	0.7804	0.7406	0.7820	0.7900
AllGesWiiY	0.7727	0.7834	0.7937	0.7651	0.8329
AllGesWiiZ	0.7661	0.6956	0.7257	0.7020	0.8114
BME	1.0000	1.0000	0.9987	1.0000	0.9933
Chinatown	0.9825	0.9738	0.9784	0.9816	0.9854
Crop	0.7513	0.7336	0.7429	0.7412	0.7722
DodgerLoopDay	0.5725	0.6800	0.1500	0.6138	0.1500
DodgerLoopGame	0.8732	0.9101	0.7101	0.9167	0.8551
DodLooWee	0.9746	0.9855	0.9522	0.9783	0.9710
EOGHorSig	0.6390	0.6210	0.5994	0.6439	0.5939
EOGVerSig	0.5414	0.5348	0.4453	0.5307	0.4751
EthanolLevel	0.5828	0.2970	0.7584	0.5656	0.8140
FreRegTra	0.9976	0.9363	0.9985	0.9980	0.9965

Table 2 continued

	Rocket	PF	ResNet	CHIEF	ITime
FreSmaTra	0.9496	0.7091	0.8322	0.9980	0.8674
Fungi	1.0000	0.9527	0.1774	1.0000	1.0000
GestureMidAirD1	0.7169	0.6723	0.6985	0.7038	0.7462
GestureMidAirD2	0.6608	0.6423	0.6677	0.6823	0.7308
GestureMidAirD3	0.4146	0.3969	0.3400	0.4046	0.4000
GesturePebbleZ1	0.9058	0.8983	0.9012	0.8866	0.9244
GesturePebbleZ2	0.8304	0.8551	0.7772	0.8209	0.8861
GunPointAgeSpan	0.9968	0.9959	0.9968	0.9946	0.9873
GunPoiMalVerFem	0.9984	0.9949	0.9924	0.9937	0.9937
GunPoiOldVerYou	0.9911	0.9832	0.9892	0.9749	0.9651
HouseTwenty	0.9639	0.9378	0.9832	0.9765	0.9748
InsEPGRegTra	1.0000	0.9703	0.9976	1.0000	1.0000
InsEPGSmaTra	0.9791	0.8703	0.3719	0.9715	0.9438
MelPed	0.9044	0.8813	0.9092	0.8817	0.9139
MixShaRegTra	0.9711	0.9624	0.9729	0.9684	0.9703
MixShaSmaTra	0.9382	0.9277	0.9165	0.9528	0.9146
PLAID	0.9026	0.8624	0.9404	0.9177	0.9441
PicGesWiiZ	0.8300	0.7120	0.7040	0.7600	0.7600
PigAirPre	0.0952	0.1793	0.4058	0.8880	0.5433
PigArtPressure	0.9538	0.6029	0.9913	0.9812	0.9952
PigCVP	0.9341	0.4418	0.9183	0.9678	0.9615
PowerCons	0.9400	0.9417	0.8789	0.9389	0.9444
Rock	0.9000	0.8660	0.5520	0.9220	0.8000
SemHanGenCh2	0.9268	0.9240	0.8237	0.8715	0.8167
SemHanMovCh2	0.6451	0.7778	0.4391	0.7624	0.4822
SemHanSubCh2	0.8811	0.9273	0.7387	0.9236	0.8244
ShaGesWiiZ	0.8980	0.8900	0.8800	0.8660	0.9000
SmoothSubspace	0.9787	1.0000	0.9800	1.0000	0.9933
UMD	0.9924	0.9903	0.9903	0.9889	0.9861

References

Bagnall A, Lines J, Bostrom A, Large J, Keogh E (2017) The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. Data Min Knowl Disc 31(3):606–660

Bagnall A, Lines J, Vickers W, Keogh E (2019) The UEA & UCR time series classification repository. http://www.timeseriesclassification.com

Bai S, Kolter JZ, Koltun V (2018) An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. arXiv:1803.01271

Benavoli A, Corani G, Mangili F (2016) Should we really use post-hoc tests based on mean-ranks? J Mach Learn Res 17(5):1–10

Bengio Y, Courville A, Vincent P (2013) Representation learning: a review and new perspectives. IEEE Trans Pattern Anal Mach Intell 35(8):1798–1828

Bostrom A, Bagnall A (2015) Binary shapelet transform for multiclass time series classification. In: Madria S, Hara T (eds) Big data analytics and knowledge discovery. Springer, Cham, pp 257–269

Bottou L, Curtis FE, Nocedal J (2018) Optimization methods for large-scale machine learning. SIAM Rev 60(2):223–311

Boureau YL, Ponce J, LeCun Y (2010) A theoretical analysis of feature pooling in visual recognition. In: Fürnkranz J, Joachims T (eds) Proceedings of the 27th international conference on machine learning, Omnipress, USA, pp 111–118

Cox D, Pinto N (2011) Beyond simple features: a large-scale feature search approach to unconstrained face recognition. Face Gesture 2011:8–15

Cui Z, Chen W, Chen Y (2016) Multi-scale convolutional neural networks for time series classification. arXiv:1603.06995

Dau HA, Bagnall A, Kamgar K, Yeh CCM, Zhu Y, Gharghabi S, Ratanamahatana CA, Keogh E (2019) The UCR time series archive. J Autom Sinica 6(6):1293–1305

Dau HA, Keogh E, Kamgar K et al (2018) UCR time series classification archive (briefing document). https://www.cs.ucr.edu/~eamonn/time_series_data_2018/

Demšar J (2006) Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res 7:1–30

Dongarra J, Gates M, Haidar A, Kurzak J, Luszczek P, Tomov S, Yamazaki I (2018) The singular value decomposition: anatomy of optimizing an algorithm for extreme scale. SIAM Rev 60(4):808–865

Farahmand A, Pourazarm S, Nikovski D (2017) Random projection filter bank for time series data. In: Guyon I, Luxburg UV, Bengio S, Wallach H, Fergus R, Vishwanathan S, Garnett R (eds) Advances in neural information processing systems, vol 30. MIT Press, Cambridge, pp 6562–6572

Franceschi J, Dieuleveut A, Jaggi M (2019) Unsupervised scalable representation learning for multivariate time series. In: Seventh international conference on learning representations, learning from limited labeled data workshop

García S, Herrera F (2008) An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. J Mach Learn Res 9:2677–2694

Goodfellow I, Bengio Y, Courville A (2016) Deep learning. MIT Press, Cambridge

Hills J, Lines J, Baranauskas E, Mapp J, Bagnall A (2014) Classification of time series by shapelet transformation. Data Min Knowl Disc 28(4):851–881

Ismail Fawaz H, Forestier G, Weber J, Idoumghar L, Muller P (2019a) Deep learning for time series classification: a review. Data Min Knowl Disc 33(4):917–963

Ismail Fawaz H, Forestier G, Weber J, Idoumghar L, Muller P (2019b) Deep neural network ensembles for time series classification. In: International joint conference on neural networks, pp 1–6

Ismail Fawaz H, Lucas B, Forestier G, Pelletier C, Schmidt DF, Weber J, Webb GI, Idoumghar L, Muller P, Petitjean F (2019c) InceptionTime: finding AlexNet for time series classification. arXiv:1909.04939

Jarrett K, Kavukcuoglu K, Ranzato M, LeCun Y (2009) What is the best multi-stage architecture for object recognition? In: 2009 IEEE 12th international conference on computer vision, pp 2146–2153

Jimenez A, Raj B (2019) Time signal classification using random convolutional features. In: 2019 IEEE international conference on acoustics, speech and signal processing

Karlsson I, Papapetrou P, Boström H (2016) Generalized random shapelet forests. Data Min Knowl Disc 30(5):1053–1085

Kingma DP, Ba JL (2015) Adam: a method for stochastic optimization. In: Third international conference on learning representations. arXiv:1412.6980

Krizhevsky A, Sutskever I, Hinton G (2012) Imagenet classification with deep convolutional neural networks. In: Pereira F, Burges CJC, Bottou L, Weinberger KQ (eds) Advances in neural information processing systems, vol 25. Curran Associates Inc, Red Hook, pp 1097–1105

Lam SK, Pitrou A, Seibert S (2015) Numba: a LLVM-based python JIT compiler. In: Proceedings of the second workshop on the LLVM compiler infrastructure in HPC, pp 1–6

Le Nguyen T, Gsponer S, Ilie I, O'Reillly M, Ifrim G (2019) Interpretable time series classification using linear models and multi-resolution multi-domain symbolic representations. Data Min Knowl Disc 33(4):1183–1222

Lin M, Chen Q, Yan S (2014) Network in network. In: Second international conference on learning representations. arXiv:1312.4400

Lines J, Taylor S, Bagnall A (2018) Time series classification with HIVE-COTE: the hierarchical vote collective of transformation-based ensembles. ACM Trans Knowl Discov Data 12(5):52:1–52:35

Lubba CH, Sethi SS, Knaute P, Schultz SR, Fulcher BD, Jones NS (2019) catch22: CAnonical Time-series CHaracteristics. Data Min Knowl Disc 33(6):1821–1852

Lucas B, Shifaz A, Pelletier C, O'Neill L, Zaidi N, Goethals B, Petitjean F, Webb GI (2019) Proximity forest: an effective and scalable distance-based classifier for time series. Data Min Knowl Disc 33(3):607–635
Middlehurst M, Vickers W, Bagnall A (2019) Scalable dictionary classifiers for time series classification. In: Yin H, Camacho D, Tino P, Tallón-Ballesteros AJ, Menezes R, Allmendinger R (eds) Intelligent data engineering and automated learning. Springer, Cham, pp 11–19
Morrow A, Shankar V, Petersohn D, Joseph A, Recht B, Yosef N (2016) Convolutional kitchen sinks for transcription factor binding site prediction. In: NIPS workshop on machine learning in computational biology
Oquab M, Bottou L, Laptev I, Sivic J (2015) Is object localization for free? Weakly-supervised learning with convolutional neural networks. In: 2015 IEEE conference on computer vision and pattern recognition, pp 685–694
Paszke A, Gross S, Chintala S, Chanan G, Yang E, DeVito Z, Lin Z, Desmaison A, Antiga L, Lerer A (2017) Automatic differentiation in PyTorch. In: NIPS autodiff workshop
Pedregosa F, Varoquaux G, Gramfort A et al (2011) Scikit-learn: machine learning in python. J Mach Learn Res 12:2825–2830
Petitjean F, Inglada J, Gancarski P (2012) Satellite image time series analysis under time warping. IEEE Trans Geosci Remote Sens 50(8):3081–3095
Pinto N, Doukhan D, DiCarlo JJ, Cox DD (2009) A high-throughput screening approach to discovering good forms of biologically inspired visual representation. PLoS Comput Biol 5(11):1–12
Rahimi A, Recht B (2008) Random features for large-scale kernel machines. In: Platt JC, Koller D, Singer Y, Roweis ST (eds) Advances in neural information processing systems, vol 20. Curran Associates Inc, Red Hook, pp 1177–1184
Rahimi A, Recht B (2009) Weighted sums of random kitchen sinks: replacing minimization with randomization in learning. In: Koller D, Schuurmans D, Bengio Y, Bottou L (eds) Advances in neural information processing systems, vol 21. MIT Press, Cambridge, pp 1313–1320
Raza A, Kramer S (2019) Accelerating pattern-based time series classification: a linear time and space string mining approach. Knowl Inf Syst 62:1113–1141
Renard X, Rifqi M, Erray W, Detyniecki M (2015) Random-shapelet: an algorithm for fast shapelet discovery. In: IEEE international conference on data science and advanced analytics, pp 1–10
Rifkin RM, Lippert RA (2007) Notes on regularized least squares. Technical report, MIT
Saxe A, Koh PW, Chen Z, Bhand M, Suresh B, Ng A (2011) On random weights and unsupervised feature learning. In: Getoor L, Scheffer T (eds) Proceedings of the 28th international conference on machine learning, Omnipress, USA, pp 1089–1096
Schäfer P (2015) The BOSS is concerned with time series classification in the presence of noise. Data Min Knowl Disc 29(6):1505–1530
Schäfer P, Leser U (2017) Fast and accurate time series classification with WEASEL. In: Proceedings of the 2017 ACM conference on information and knowledge management, pp 637–646
Shifaz A, Pelletier C, Petitjean F, Webb GI (2020) TS-CHIEF: a scalable and accurate forest algorithm for time series classification. Data Min Knowl Discov 34:742–775
Wang Z, Yan W, Oates T (2017) Time series classification from scratch with deep neural networks: a strong baseline. In: 2017 international joint conference on neural networks, pp 1578–1585
Wistuba M, Grabocka J, Schmidt-Thieme L (2015) Ultra-fast shapelets for time series classification. arXiv:1503.05018
Yosinski J, Clune J, Bengio Y, Lipson H (2014) How transferable are features in deep neural networks? In: Ghahramani Z, Welling M, Cortes C, Lawrence ND, Weinberger KQ (eds) Advances in neural information processing systems, vol 27. MIT Press, Cambridge, pp 3320–3328
Yu F, Koltun V (2016) Multi-scale context aggregation by dilated convolutions. In: Fourth international conference on learning representations. arXiv:1511.07122
Zeiler MD, Fergus R (2014) Visualizing and understanding convolutional networks. In: Fleet D, Pajdla T, Schiele B, Tuytelaars T (eds) European conference on computer vision. Springer, Cham, pp 818–833