

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/336889756>

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

Preprint · October 2019

CITATIONS

0

READS

164

3 authors, including:



[François Petitjean](#)

Monash University (Australia)

82 PUBLICATIONS 1,195 CITATIONS

[SEE PROFILE](#)



[Geoffrey I Webb](#)

Monash University (Australia)

409 PUBLICATIONS 8,251 CITATIONS

[SEE PROFILE](#)

Some of the authors of this publication are also working on these related projects:



Protease systems biology [View project](#)



Graphical Models [View project](#)

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

Angus Dempster · François Petitjean ·  
Geoffrey I. Webb

Received: date / Accepted: date

**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.

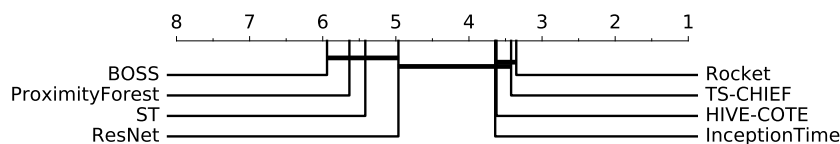
**Keywords** scalable · time series classification · random · convolution

## 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 (Shifaz et al. 2019), and InceptionTime (Ismail Fawaz et al. 2019c).

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 **K**ernel **T**ransform).

Angus Dempster · François Petitjean · Geoffrey I. Webb  
Faculty of Information Technology, Monash University, Melbourne, Australia  
E-mail: {angus.dempster1,francois.petitjean,geoff.webb}@monash.edu



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

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 specialized 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). Restricted to a single CPU core, the total training time for ROCKET is:

- 6 minutes for the ‘bake off’ dataset with the largest training set (*ElectricDevices*, with 8,926 training examples), compared to 1 hour 35 minutes for Proximity Forest, 2 hours 24 minutes for TS-CHIEF, and 7 hours 46 minutes for InceptionTime (trained on GPUs); and
- 4 minutes and 52 seconds for the ‘bake off’ dataset with the longest time series (*HandOutlines*, with time series of length 2,709), compared to 8 hours 10 minutes 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 hour 50 minutes, compared to 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.)

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 hour 15 minutes, to a similar accuracy as Proximity Forest, which requires more than 16 hours 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 minute, 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 section 2, we review relevant related work. In section 3, we explain ROCKET in detail. In section 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), 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,  $O(n^2 \cdot l^2)$ . BOSS-VS is a more scalable variant of BOSS, but is less accurate (Schäfer 2016). 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 (Lines et al. 2018). 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  $O(n^2 \cdot l^2)$  (Lines et al. 2018).

### 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, and InceptionTime.

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. 2019). Like Proximity Forest, TS-

CHIEF has a training complexity quasilinear in the number of training examples, but quadratic 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](#), p. 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 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 and 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. 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](#), 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. \(2017\)](#) 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. \(2017](#), 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. (In both cases, there are some differences with the convolution operation as used in typical convolutional neural networks.) [Farahmand et al. \(2017\)](#) propose a feature transformation based on convolving input time series with random autoregressive filters.

A number of things distinguish ROCKET from convolutional layers as used in typical convolutional neural networks, and from other methods using convolutional kernels (including random convolutional kernels) in relation to time series, set out in detail in section 3. We show that leveraging all aspects of kernel architecture—crucially, with a variety of random length, dilation, and padding (as well as weights and bias), and drawing an effective set of features from the output of the convolutions—provides for state-of-the-art accuracy with a fraction of the computational expense of existing state-of-the-art methods.

### 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. 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. However, in practice, for all but the largest datasets, we use a ridge regression classifier, which has the advantage of fast cross-validation for the regularization 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 computational 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 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 section 4.3.4, below).
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 section 4.3.6).

In effect, the only hyperparameter for ROCKET is the number of kernels,  $k$ . 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 section 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 (we use 10,000 by default), ROCKET is extremely fast.

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 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. 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,  $\forall w \in \mathbf{W}$ ,  $w \sim \mathcal{N}(0, 1)$ , and are mean centered after being set,  $\omega = \mathbf{W} - \overline{\mathbf{W}}$ . 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 used (see section 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,  $l_{\text{input}}$ . 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 section 3.4.1).

Stride is always one. We do not apply a nonlinearity such as ReLU to the resulting feature maps (indeed both *ppv* and *max* are agnostic to ReLU). 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 normalized to have a mean of zero and a standard deviation of one (see generally [Dau et al. 2019](#)).

As noted above, these parameters were determined to produce the highest classification accuracy on the ‘development’ datasets. However, as demonstrated in section 4.3, below, there are several alternative configurations which produce similar classification accuracy. 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,  $\omega$ , with dilation,  $d$ , to a given time series,  $X$ , from position  $i$  in  $X$ , is given by (see, e.g., [Bai et al. 2018](#)):

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

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*).

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 feature computed by ROCKET on each feature map is *ppv*. The *ppv* directly captures the proportion of the input which matches a given pattern. We found that *ppv* produces meaningfully higher classification accuracy than other features, including the mean (broadly equivalent to global average pooling).

For  $k$  kernels, ROCKET produces  $2k$  features per time series (i.e., *ppv* and *max*). For 10,000 kernels (the default), ROCKET produces 20,000 features. For smaller datasets (in fact, for all 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).

*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. (A ridge regression model is trained for each class in a ‘one versus rest’ fashion, with  $L_2$  regularization.)

Regularization is critically important where the number of features is significantly greater than the number of training examples, allowing for the optimization 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 regularization 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.

### 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  $l_{\text{input}}$  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  $l_{\text{input}}$ , 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  $l_{\text{input}}$ , or worst-case complexity for a maximum length of  $l_{\text{input}}$ .

### 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 generalized cross-validation to determine appropriate regularization. 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 regularization 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.

## 4 Experiments

We evaluate ROCKET on the UCR archive (section 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 (section 4.2), demonstrating that ROCKET is orders of magnitude faster than current methods. We also evaluate the effect of different kernel parameters (section 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

#### 4.1.1 ‘Bake Off’ Datasets

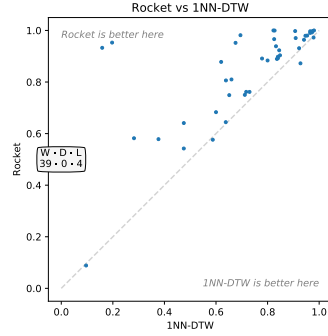
We evaluate ROCKET on the 85 ‘bake off’ datasets from the UCR archive (on the original training/test split for each dataset). The results presented for ROCKET are mean results over 10 runs (using a different set of random kernels for each run).

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 (with papers on arXiv), 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, ROCKET was developed using a subset of 40 randomly-selected datasets, to make sure we didn’t overfit the UCR archive. 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 Figure 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-



**Fig. 2** Relative accuracy of ROCKET versus 1NN-DTW on the 43 additional 2018 datasets.

CHIEF is not statistically significant. TS-CHIEF ranks ahead of ROCKET on the 45 ‘holdout’ datasets (Figure 14, Appendix B), but the difference is not significant.

#### 4.1.2 Additional 2018 Datasets

We have also evaluated ROCKET on the 43 additional datasets in the UCR archive as of 2018, in order to: (1) show that our method is able to handle datasets with varying lengths; and (2) provide reference results for future research papers.

There are no published results for state-of-the-art methods on these datasets. Adapting these methods to work on variable-length time series is nontrivial, as the most appropriate method for handling variable lengths (1) depends on whether the variable lengths represent subsampling or variable sampling frequencies, and (2) is classifier dependent (Tan et al. 2019). Accordingly, we restrict our comparison to the available results for 1NN-DTW (Dau et al. 2019), where 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). Figure 2 shows the relative accuracy of ROCKET and 1NN-DTW on the 43 additional datasets. ROCKET is more accurate on all but four datasets, and substantially more so on most.

Following Dau et al. (2018), we have normalized each time series and interpolated missing values. Variable-length time series have been rescaled or used ‘as is’ (with their original lengths) as determined by 10-fold cross-validation.

## 4.2 Scalability

### 4.2.1 Training Set Size

Following Lucas et al. (2019), Shifaz et al. (2019), 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.



**Fig. 3** Accuracy (left) and training time (right) versus training set size.

For this purpose, we integrate ROCKET with logistic regression. The transform is performed in tranches, which are further divided into minibatches for training. Each time series is normalised to have a zero mean and unit standard deviation.

We train the model for at least one epoch for each subset 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 2,048 examples for all subset sizes).

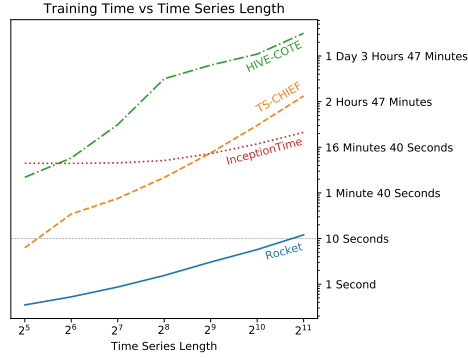
Optimization is performed using Adam (Kingma and Ba 2015). We perform a minimal search on the initial learning rate to ensure that training loss does not diverge. The learning rate is halved if training loss fails to improve after 100 updates (only relevant for larger subset sizes).

We have run ROCKET in three guises: with 100, 1,000, 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. 2019). (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 section 3.4). With 1,000 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.)

#### 4.2.2 Time Series Length

Following Shifaz et al. (2019) 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



**Fig. 4** Training time versus time series length.

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. \(2019\)](#).

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 2,048 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.

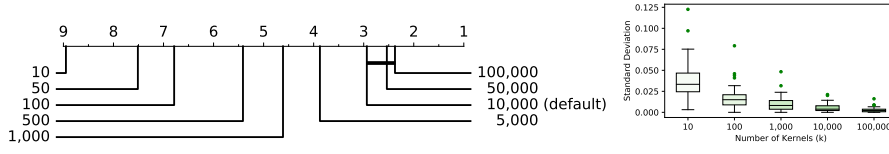
#### 4.3 Sensitivity Analysis

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 section 3) against different choices for the number of kernels, length, weights and bias, dilation, padding, and output features. 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).

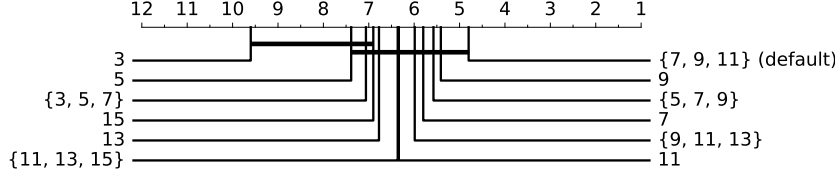
In most cases, alternative configurations represent a relatively subtle change from the default configuration. Unsurprisingly, therefore, in many cases one or more alternative choices for the relevant parameter produces similar accuracy to the baseline configuration. In other words, ROCKET is relatively 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

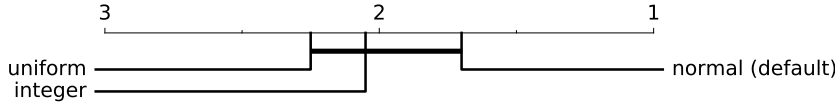
We evaluate increasing numbers of kernels between 10 and 100,000. Figure 5 shows the effect of the number of kernels,  $k$ , on accuracy. Clearly, increasing the number of kernels improves accuracy. However, the actual difference in accuracy between, for example,  $k = 5,000$  and  $k = 10,000$ , is relatively small, even if statistically



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



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



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

significant. Indeed,  $k = 5,000$  produces higher accuracy on some datasets (Figure 16, Appendix C). Nevertheless,  $k = 10,000$  is noticeably ahead in terms of win/draw/loss (29/3/8). The differences between  $k = 10,000$ ,  $k = 50,000$ , and  $k = 100,000$  are not statistically significant.

Even though ROCKET is nondeterministic, the variability in accuracy is reasonably low for large numbers of kernels. Unsurprisingly, standard deviation diminishes as  $k$  increases. The median standard deviation across the 40 ‘development’ datasets is 0.0038 for  $k = 10,000$ , and 0.0021 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}.

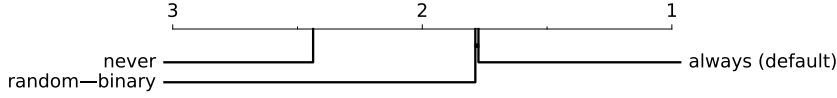
Figure 6 shows the effect of these choices on accuracy. Fixed lengths of 7, 9, and 11, as well as selecting length randomly from {5, 7, 9} and {9, 11, 13} result in similar accuracy to the default configuration, and the differences are not statistically significant (see also Figure 17, Appendix C). Shorter kernels are undesirable, being more strongly correlated with each other for a large number of kernels.

#### 4.3.3 Weights (Including Centering) and Bias

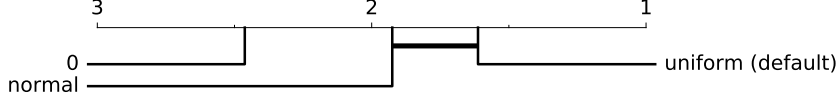
*Weights.* We vary the distribution from which the weights are sampled, comparing the baseline (sampling from a normal distribution) to:

- sampling from a uniform distribution,  $\forall w \in \mathbf{W}, w \sim \mathcal{U}(-1, 1)$ ; and

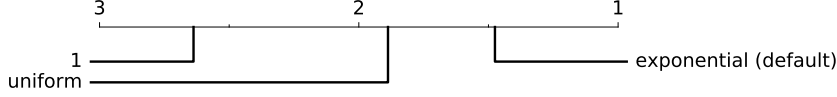




**Fig. 8** Mean ranks for different choices in terms of centering.



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



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

- 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 Figure 18, 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 (Figure 19, Appendix C).

*Bias.* We vary bias, comparing the baseline (using a uniform distribution) against:

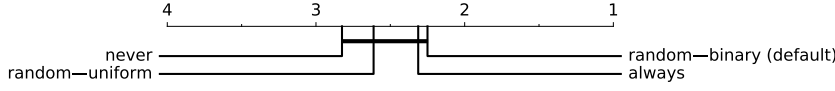
- using zero bias; and
- sampling bias from a normal distribution,  $b \sim \mathcal{N}(0, 1)$ .

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 Figure 20, Appendix C.)

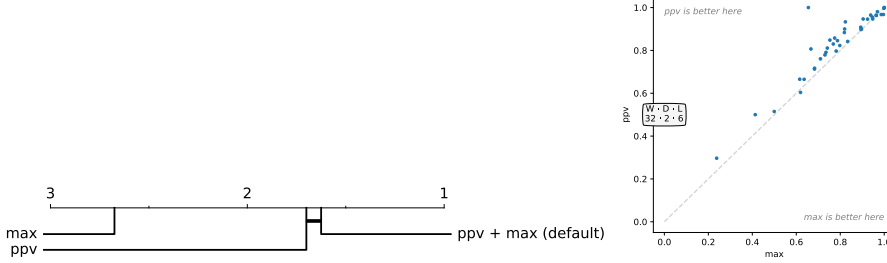
#### 4.3.4 Dilation

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

- no dilation (i.e., a fixed dilation of one); and



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



**Fig. 12** Mean ranks (left), and relative accuracy (right), *ppv* and *max*.

- 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 Figure 21, 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 (Figure 22, Appendix C).

#### 4.3.6 Features

We vary the output features, comparing the baseline, *ppv* and *max*, against using each in isolation. Figure 12 shows the effect of these choices on accuracy. It is clear that *ppv* is superior to *max*: *ppv* produces substantially higher classification accuracy for the majority of the ‘development’ datasets. In fact, *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 Figure 23, 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 methods. ROCKET also scales to millions of time series.

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 timeseries, 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. \(2019a\)](#).

## References

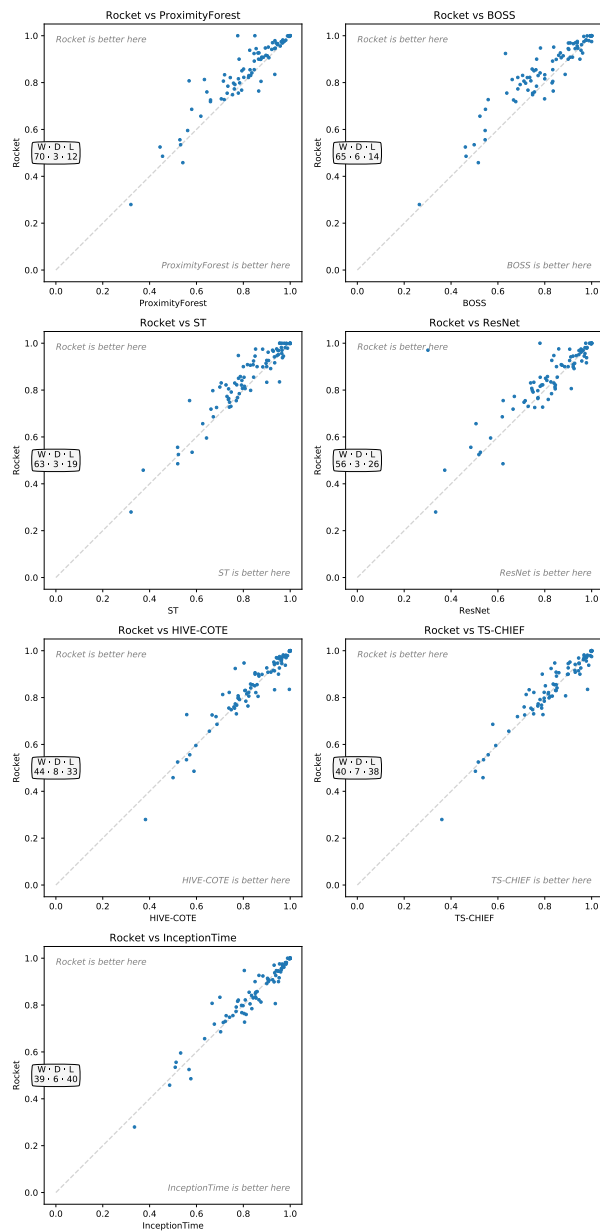
- Bagnall A, Lines J, Bostrom A, Large J, Keogh E (2017) The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery* 31(3):606–660
- Bagnall A, Lines J, Vickers W, Keogh E (2019) The UEA & UCR time series classification 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:180301271*
- Benavoli A, Corani G, Mangili F (2016) Should we really use post-hoc tests based on mean-ranks? *Journal of Machine Learning Research* 17(5):1–10
- Bengio Y, Courville A, Vincent P (2013) Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 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 Review* 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. In: *Face and Gesture 2011*, pp 8–15

- 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/](https://www.cs.ucr.edu/~eamonn/time_series_data_2018/)
- Dau HA, Bagnall A, Kamgar K, Yeh CCM, Zhu Y, Gharghabi S, Ratanamahatana CA, Keogh E (2019) The UCR time series archive. arXiv:181007758
- Demšar J (2006) Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research* 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 Review* 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* 30, 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. *Journal of Machine Learning Research* 9:2677–2694
- Goodfellow I, Bengio Y, Courville A (2016) *Deep Learning*. MIT Press, Cambridge, MA
- Hills J, Lines J, Baranauskas E, Mapp J, Bagnall A (2014) Classification of time series by shapelet transformation. *Data Mining and Knowledge Discovery* 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 Mining and Knowledge Discovery* 33(4):917–963
- Ismail Fawaz H, Forestier G, Weber J, Idoumghar L, Muller P (2019b) Deep neural network ensembles for time series classification. arXiv:190306602
- 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:190904939
- 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*
- 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* 25, 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
- 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 Transactions on Knowledge Discovery from Data* 12(5):52:1–52:35
- 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 Mining and Knowledge Discovery* 33(3):607–635
- Morrow A, Shankar V, Petersohn D, Joseph A, Recht B, Yosef N (2017) Convolutional kitchen sinks for transcription factor binding site prediction. arXiv:170600125
- 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. *Journal of Machine Learning Research* 12:2825–2830
- Petitjean F, Inglada J, Gancarski P (2012) Satellite image time series analysis under time warping. *IEEE Transactions on Geoscience and Remote Sensing* 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 Computational*

- Biology 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* 20, 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* 21, pp 1313–1320
- Rifkin RM, Lippert RA (2007) Notes on regularized least squares. Tech. rep., 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 Mining and Knowledge Discovery* 29(6):1505–1530
- Schäfer P (2016) Scalable time series classification. *Data Mining and Knowledge Discovery* 30(5):1273–1298
- 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 (2019) TS-CHIEF: A scalable and accurate forest algorithm for time series classification. [arXiv:1906.10329](https://arxiv.org/abs/1906.10329)
- Tan CW, Petitjean F, Keogh E, Webb GI (2019) Time series classification for varying length series. [arXiv:1910.04341](https://arxiv.org/abs/1910.04341)
- 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
- 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* 27, 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](https://arxiv.org/abs/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

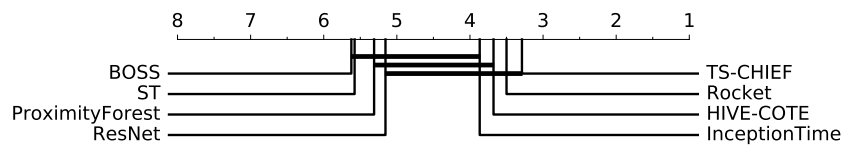
## Appendices

### A Relative Accuracy

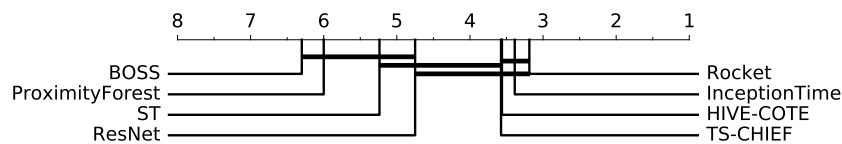


**Fig. 13** Relative accuracy of ROCKET vs state-of-the-art classifiers on the ‘bake off’ datasets.

## B ‘Development’ and ‘Holdout’ Datasets

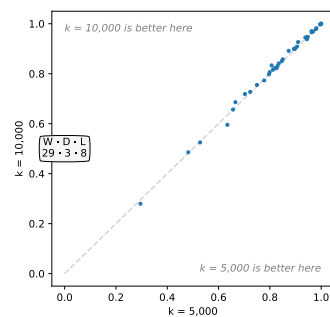


**Fig. 14** Mean rank of ROCKET versus state-of-the-art classifiers on the ‘holdout’ datasets.

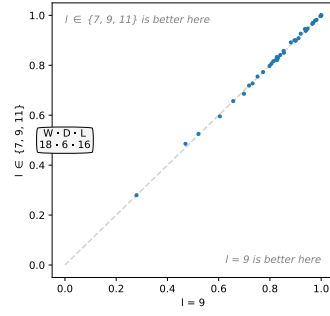


**Fig. 15** Mean rank of ROCKET vs state-of-the-art classifiers on the ‘development’ datasets.

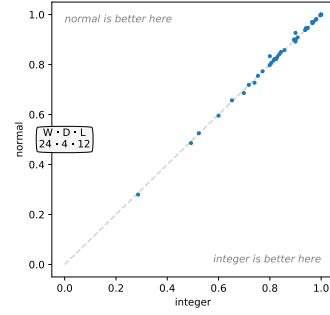
## C Additional Plots



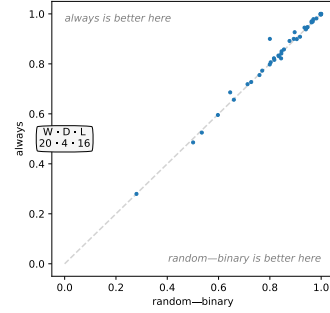
**Fig. 16** Relative accuracy of  $k = 10,000$  versus  $k = 5,000$  on the ‘development’ datasets.



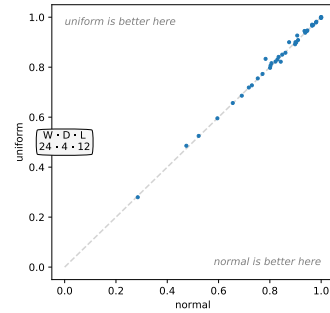
**Fig. 17** Relative accuracy of  $l \in \{7, 9, 11\}$  versus  $l = 9$  on the ‘development’ datasets.



**Fig. 18** Relative accuracy, normally-distributed vs integer weights, ‘development’ datasets.

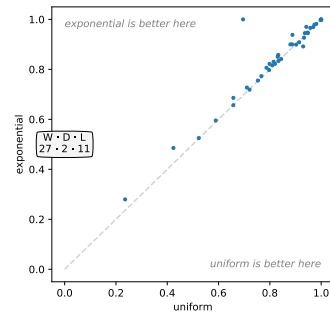


**Fig. 19** Relative accuracy of always vs random centering on the ‘development’ datasets.

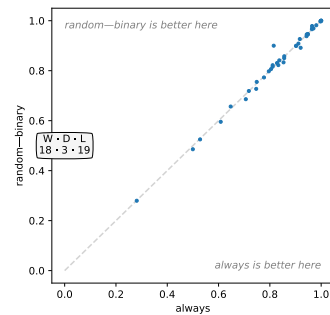


**Fig. 20** Relative accuracy, uniformly versus normally-distributed bias, ‘development’ datasets.

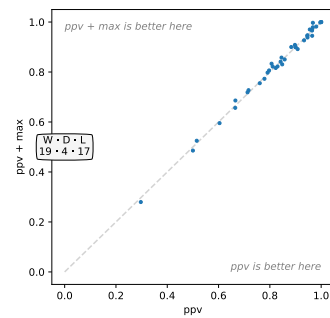




**Fig. 21** Relative accuracy of exponential vs uniform dilation on the ‘development’ datasets.



**Fig. 22** Relative accuracy of random versus always padding on the ‘development’ datasets.



**Fig. 23** Relative accuracy of *ppv* and *max* versus only *ppv* on the ‘development’ datasets.

## D Results for ‘Bake Off’ Datasets

The ‘development’ datasets are marked with an asterisk.

Table 1: Classification Accuracy, ‘Bake Off’ Datasets

	Rocket	BOSS	ST	HCTE	ResNet	PF	CHIEF	ITime
Adiac	0.7847	0.7647	0.7826	0.8107	0.8289	0.7340	0.7980	0.8363
ArrowHead	0.8051	0.8343	0.7371	0.8629	0.8446	0.8754	0.8229	0.8286
*Beef	0.8333	0.8000	0.9000	0.9333	0.7533	0.7200	0.7333	0.7000
BeetleFly	0.9000	0.9000	0.9000	0.9500	0.8500	0.8750	0.9500	0.8500
*BirdChicken	0.9000	0.9500	0.8000	0.8500	0.8850	0.8650	0.9000	0.9500
CBF	0.9999	0.9978	0.9744	0.9989	0.9950	0.9933	0.9978	0.9989
*Car	0.8917	0.8333	0.9167	0.8667	0.9250	0.8467	0.8500	0.9000
ChlCon	0.8130	0.6609	0.6997	0.7120	0.8436	0.6339	0.7206	0.8753
CinCECGTorso	0.8349	0.8870	0.9543	0.9964	0.8261	0.9343	0.9826	0.8514
Coffee	1.0000	1.0000	0.9643	1.0000	1.0000	1.0000	1.0000	1.0000
Computers	0.7600	0.7560	0.7360	0.7600	0.8148	0.6444	0.7120	0.8120
*CricketX	0.8223	0.7359	0.7718	0.8231	0.7913	0.8021	0.7974	0.8667
*CricketY	0.8503	0.7538	0.7795	0.8487	0.8033	0.7938	0.8026	0.8513
*CricketZ	0.8577	0.7462	0.7872	0.8308	0.8115	0.8010	0.8359	0.8590
DiaSizRed	0.9703	0.9314	0.9248	0.9412	0.3013	0.9657	0.9771	0.9314
DisPhaOutAgeGro	0.7547	0.7482	0.7698	0.7626	0.7165	0.7309	0.7410	0.7266
DisPhaOutCor	0.7678	0.7283	0.7754	0.7717	0.7710	0.7928	0.7862	0.7935
*DisPhaTW	0.7187	0.6763	0.6619	0.6835	0.6647	0.6597	0.6835	0.6763
ECG200	0.9060	0.8700	0.8300	0.8500	0.8740	0.9090	0.8600	0.9100
*ECG5000	0.9470	0.9413	0.9438	0.9462	0.9342	0.9365	0.9458	0.9409
ECGFiveDays	1.0000	1.0000	0.9837	1.0000	0.9748	0.8492	1.0000	1.0000
Earthquakes	0.7482	0.7482	0.7410	0.7482	0.7115	0.7540	0.7482	0.7410
ElectricDevices	0.7305	0.7992	0.7470	0.7703	0.7291	0.7060	0.7524	0.7227
FaceAll	0.9475	0.7817	0.7787	0.8030	0.8388	0.8938	0.8426	0.8041
FaceFour	0.9750	1.0000	0.8523	0.9545	0.9545	0.9739	1.0000	0.9659
FacesUCR	0.9616	0.9571	0.9059	0.9629	0.9547	0.9459	0.9649	0.9732
*FiftyWords	0.8305	0.7055	0.7055	0.8088	0.7396	0.8314	0.8462	0.8418
*Fish	0.9789	0.9886	0.9886	0.9886	0.9794	0.9349	0.9943	0.9829
*FordA	0.9449	0.9295	0.9712	0.9644	0.9205	0.8546	0.9470	0.9483
*FordB	0.8063	0.7111	0.8074	0.8235	0.9131	0.7149	0.8321	0.9365
GunPoint	1.0000	1.0000	1.0000	1.0000	0.9907	0.9973	1.0000	1.0000
Ham	0.7257	0.6667	0.6857	0.6667	0.7571	0.6600	0.7143	0.7143
HandOutlines	0.9416	0.9027	0.9324	0.9324	0.9111	0.9214	0.9297	0.9595
*Haptics	0.5250	0.4610	0.5227	0.5195	0.5188	0.4445	0.5162	0.5682
*Herring	0.6859	0.5469	0.6719	0.6875	0.6188	0.5797	0.5781	0.7031
InlineSkate	0.4582	0.5164	0.3727	0.5000	0.3731	0.5418	0.5364	0.4855
*InsWinSou	0.6566	0.5232	0.6268	0.6551	0.5065	0.6187	0.6465	0.6348
*ItaPowDem	0.9691	0.9086	0.9475	0.9631	0.9630	0.9671	0.9718	0.9679
*LarKitApp	0.9000	0.7653	0.8587	0.8640	0.8997	0.7819	0.7893	0.9067
Lightning2	0.7639	0.8361	0.7377	0.8197	0.7705	0.8656	0.7705	0.8033
*Lightning7	0.8219	0.6849	0.7260	0.7397	0.8452	0.8219	0.7534	0.8082
Mallat	0.9560	0.9382	0.9642	0.9620	0.9716	0.9576	0.9774	0.9629
*Meat	0.9450	0.9000	0.8500	0.9333	0.9683	0.9333	0.9000	0.9500
*MedicalImages	0.7975	0.7184	0.6697	0.7776	0.7703	0.7582	0.7974	0.7987
*MidPhaOutAgeGro	0.5955	0.5455	0.6429	0.5974	0.5688	0.5623	0.5909	0.5325
*MidPhaOutCor	0.8412	0.7801	0.7938	0.8316	0.8089	0.8364	0.8522	0.8351
MiddlePhalanxTW	0.5558	0.5455	0.5195	0.5714	0.4844	0.5292	0.5584	0.5130
MoteStrain	0.9142	0.8786	0.8970	0.9329	0.9276	0.9024	0.9441	0.9034
NonInvFetECGTho1	0.9514	0.8382	0.9496	0.9303	0.9454	0.9066	0.9074	0.9623
NonInvFetECGTho2	0.9688	0.9008	0.9511	0.9445	0.9461	0.9399	0.9445	0.9674
*OSULeaf	0.9380	0.9545	0.9669	0.9793	0.9785	0.8273	0.9876	0.9339
*OliveOil	0.9267	0.8667	0.9000	0.9000	0.8300	0.8667	0.9000	0.8667

Table 1: Classification Accuracy, ‘Bake Off’ Datasets

	Rocket	BOSS	ST	HCTE	ResNet	PF	CHIEF	ITime
PhaOutCor	0.8300	0.7716	0.7634	0.8065	0.8390	0.8235	0.8485	0.8543
*Phoneme	0.2796	0.2648	0.3207	0.3824	0.3343	0.3201	0.3608	0.3354
*Plane	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
ProPhaOutAgeGro	0.8551	0.8341	0.8439	0.8585	0.8532	0.8463	0.8488	0.8537
*ProPhaOutCor	0.8990	0.8488	0.8832	0.8797	0.9213	0.8732	0.8969	0.9313
*ProPhaTW	0.8161	0.8000	0.8049	0.8146	0.7805	0.7790	0.8146	0.7756
RefDev	0.5347	0.4987	0.5813	0.5573	0.5253	0.5323	0.5387	0.5093
*ScreenType	0.4856	0.4640	0.5200	0.5893	0.6216	0.4552	0.5040	0.5760
*ShapeletSim	1.0000	1.0000	0.9556	1.0000	0.7794	0.7761	1.0000	0.9889
ShapesAll	0.9082	0.9083	0.8417	0.9050	0.9213	0.8858	0.9300	0.9250
SmaKitApp	0.8213	0.7253	0.7920	0.8533	0.7861	0.7443	0.8160	0.7787
SonAIBORobSur1	0.9241	0.6323	0.8436	0.7654	0.9581	0.8458	0.8270	0.8835
SonAIBORobSur2	0.9164	0.8594	0.9339	0.9276	0.9778	0.8963	0.9286	0.9528
StarLightCurves	0.9811	0.9778	0.9785	0.9815	0.9718	0.9813	0.9820	0.9792
*Strawberry	0.9819	0.9757	0.9622	0.9703	0.9805	0.9684	0.9676	0.9838
*SwedishLeaf	0.9659	0.9216	0.9280	0.9536	0.9563	0.9466	0.9664	0.9712
Symbols	0.9746	0.9668	0.8824	0.9739	0.9064	0.9616	0.9799	0.9819
*SynCon	0.9970	0.9667	0.9833	0.9967	0.9983	0.9953	1.0000	0.9967
*ToeSeg1	0.9702	0.9386	0.9649	0.9825	0.9627	0.9246	0.9693	0.9693
ToeSeg2	0.9262	0.9615	0.9077	0.9538	0.9062	0.8623	0.9538	0.9385
*Trace	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
TwoLeadECG	0.9991	0.9807	0.9974	0.9965	1.0000	0.9886	0.9965	0.9956
TwoPatterns	1.0000	0.9930	0.9550	1.0000	0.9999	0.9996	1.0000	1.0000
UWavGesLibAll	0.9757	0.9389	0.9422	0.9685	0.8595	0.9723	0.9687	0.9545
UWavGesLibX	0.8546	0.7621	0.8029	0.8398	0.7805	0.8286	0.8417	0.8247
*UWavGesLibY	0.7729	0.6851	0.7303	0.7655	0.6701	0.7615	0.7716	0.7688
UWavGesLibZ	0.7917	0.6949	0.7485	0.7831	0.7501	0.7640	0.7797	0.7697
*Wafer	0.9983	0.9948	1.0000	0.9994	0.9986	0.9955	0.9989	0.9987
Wine	0.8074	0.7407	0.7963	0.7778	0.7444	0.5685	0.8889	0.6667
*WordSynonyms	0.7552	0.6379	0.5705	0.7382	0.6224	0.7787	0.7868	0.7555
*Worms	0.7273	0.5584	0.7403	0.5584	0.7909	0.7182	0.7922	0.8052
WormsTwoClass	0.7987	0.8312	0.8312	0.7792	0.7468	0.7844	0.8182	0.7922
*Yoga	0.9085	0.9183	0.8177	0.9177	0.8702	0.8786	0.8483	0.9057

## E Results for Additional 2018 Datasets

Table 2: Classification Accuracy, Additional 2018 Datasets

	Rocket	DTW
ACSF1	0.8780	0.6200
AllGestureWiimoteX	0.7619	0.7171
AllGestureWiimoteY	0.7617	0.7300
AllGestureWiimoteZ	0.7491	0.6514
BME	1.0000	0.9800
Chinatown	0.9802	0.9536
Crop	0.7502	0.7117
DodgerLoopDay	0.5762	0.5875
DodgerLoopGame	0.8725	0.9275
DodgerLoopWeekend	0.9725	0.9783
EOGHorizontalSignal	0.6409	0.4751
EOGVerticalSignal	0.5423	0.4751
EthanolLevel	0.5820	0.2820
FreezerRegularTrain	0.9976	0.9070
FreezerSmallTrain	0.9519	0.6758
Fungi	1.0000	0.8226
GestureMidAirD1	0.8062	0.6385
GestureMidAirD2	0.6831	0.6000
GestureMidAirD3	0.5785	0.3769
GesturePebbleZ1	0.9663	0.8256
GesturePebbleZ2	0.8911	0.7785
GunPointAgeSpan	0.9968	0.9652
GunPointMaleVersusFemale	0.9978	0.9747
GunPointOldVersusYoung	0.9905	0.9651
HouseTwenty	0.9639	0.9412
InsectEPGRegularTrain	0.9996	0.8273
InsectEPGSmallTrain	0.9815	0.6948
MelbournePedestrian	0.9035	0.8482
MixedShapesRegularTrain	0.9704	0.9089
MixedShapesSmallTrain	0.9386	0.8326
PLAID	0.8896	0.8361
PickupGestureWiimoteZ	0.8100	0.6600
PigAirwayPressure	0.0885	0.0962
PigArtPressure	0.9529	0.1971
PigCVP	0.9327	0.1587
PowerCons	0.9311	0.9222
Rock	0.8980	0.8400
SemgHandGenderCh2	0.9230	0.8450
SemgHandMovementCh2	0.6444	0.6378
SemgHandSubjectCh2	0.8836	0.8000
ShakeGestureWiimoteZ	0.8920	0.8400
SmoothSubspace	0.9793	0.9467
UMD	0.9924	0.9722