MiniRocket

A Very Fast (Almost) Deterministic Transform for Time Series Classification

Angus Dempster
angus.dempster1@monash.edu
Monash University
Melbourne, Australia

Daniel F. Schmidt
daniel.schmidt@monash.edu
Monash University
Melbourne, Australia

Geoffrey I. Webb
geoff.webb@monash.edu
Monash University
Melbourne, Australia

ABSTRACT

Rocket achieves state-of-the-art accuracy for time series classification with a fraction of the computational expense of most existing methods by transforming input time series using random convolutional kernels, and using the transformed features to train a linear classifier. We reformulate Rocket into a new method, MiniRocket. MiniRocket is up to 75 times faster than Rocket on larger datasets, and almost deterministic (and optionally, fully deterministic), while maintaining essentially the same accuracy. Using this method, it is possible to train and test a classifier on all of 109 datasets from the UCR archive to state-of-the-art accuracy in under 10 minutes. MiniRocket is significantly faster than any other method of comparable accuracy (including Rocket), and significantly more accurate than any other method of remotely similar computational expense.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; • **Information systems** → **Data mining**.

KEYWORDS

scalable; time series classification; convolution; transform

ACM Reference Format:
Angus Dempster, Daniel F. Schmidt, and Geoffrey I. Webb. 2021. MiniRocket: A Very Fast (Almost) Deterministic Transform for Time Series Classification. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '21), August 14–18, 2021, Virtual Event, Singapore.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3447548.3467231

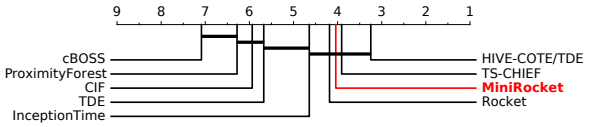

Figure 1: Mean rank of MiniRocket in terms of accuracy versus other SOTA methods over 30 resamples of 109 datasets from the UCR archive.

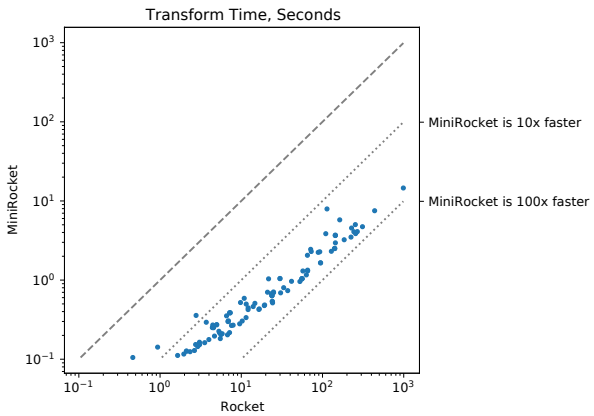

Figure 2: Transform time for MiniRocket versus Rocket for the same 109 datasets from the UCR archive.

1 INTRODUCTION

Until recently, the most accurate methods for time series classification were limited by high computational complexity. While there have been considerable advances in recent years, computational complexity and a lack of scalability remain persistent problems.

Rocket [8] achieves state-of-the-art accuracy with a fraction of the computational expense of any method of comparable accuracy by transforming input time series using random convolutional kernels, and using the transformed features to train a linear classifier. We show that it is possible to reformulate Rocket, making it up to 75 times faster on larger datasets, and making it almost entirely deterministic (and optionally, with additional computational expense, fully deterministic), while maintaining essentially the same accuracy. We call this method MiniRocket (for **MINI**mally **R**and**O**m **C**onvolutional **KE**rnel **T**ransform).

Like Rocket, MiniRocket transforms input time series using convolutional kernels, and uses the transformed features to train a linear classifier. However, unlike Rocket, MiniRocket uses a small, fixed set of kernels, and is almost entirely deterministic. MiniRocket maintains the two most important aspects of Rocket: dilation and PPV, i.e., 'proportion of positive values' pooling [8]. MiniRocket exploits various properties of the kernels, and of PPV, in order to massively reduce the time required for the transform. MiniRocket demonstrates that, while random convolutional kernels are highly effective, it is possible to achieve essentially the same accuracy using a mostly-deterministic and much faster procedure.

Figure 1 shows the mean rank of MiniRocket in terms of accuracy versus other state-of-the-art methods over 30 resamples of 109

KDD '21, August 14–18, 2021, Virtual Event, Singapore

ACM ISBN 978-1-4503-8332-5/21/08…$15.00
https://doi.org/10.1145/3447548.3467231

datasets from the UCR archive of benchmark time series [7]. On average, MiniRocket is marginally more accurate than Rocket, and slightly less accurate than the most accurate current methods.

Figure 2 shows total transform time (training and test) for MiniRocket versus Rocket, for the same 109 datasets. On average, MiniRocket is more than 30 times faster than Rocket (the advantage is even greater for larger datasets: see Section 4.2). Restricted to a single CPU core, total compute time for Rocket over all 109 datasets is 2 hours 2 minutes (1h 55m transform time), versus just 8 minutes for MiniRocket (2m 30s transform time). To put this in context, total compute time for MiniRocket for all 109 datasets is less than the compute time for Rocket for just one of those datasets. (Compute times are averages over 30 resamples, run on a cluster using Intel Xeon E5-2680 v3/4 and Xeon Gold 6150 CPUs, restricted to a single CPU core per dataset per resample.)

While only broadly comparable due to hardware and software differences, total compute time for the same 109 datasets using a single CPU thread is approximately 13 hours for cBOSS, more than a day for CIF, more than two days for TDE, approximately a week for Proximity Forest, more than two weeks for HIVE-COTE, and several weeks for TS-CHIEF [2, 27, 28]. Total compute time for InceptionTime (using GPUs, and for the original training/test splits rather than resamples) is more than 4 days [16].

MiniRocket represents a significant advance in accuracy relative to computational cost. MiniRocket is significantly faster than any other method of comparable accuracy (including Rocket), and significantly more accurate than any other method of even roughly-similar computational expense.

The rest of this paper is structured as follows. In Section 2, we review relevant related work. In Section 3, we detail the changes from Rocket to MiniRocket. In Section 4, we present experimental results for MiniRocket in terms of accuracy and scalability, as well as a sensitivity analysis in relation to key parameter choices.

2 RELATED WORK

2.1 Current State of the Art

Recent advances in accuracy have largely superseded the most accurate methods originally identified in [3]. According to [2, 27, 28], the most accurate current methods for time series classification are HIVE-COTE and its variants [22], TS-CHIEF [36], InceptionTime [16], and Rocket [8]. However, while accuracy has improved, with some exceptions computational complexity and a lack of scalability remain persistent problems.

TS-CHIEF builds on Proximity Forest, an ensemble of decision trees using distance measures as splitting criteria [25]. In addition to distance measures, TS-CHIEF uses interval-based and spectral-based splitting criteria [36].

InceptionTime is an ensemble of convolutional neural networks based on the Inception architecture, and is the most accurate convolutional neural network model for time series classification [16].

The Temporal Dictionary Ensemble (TDE) is a recent dictionary method based on the frequency of occurrence of patterns in time series [28]. TDE combines aspects of earlier dictionary methods including cBOSS [29], a more scalable variant of BOSS [34].

Catch22 is a transform based on 22 predefined time series features, used in combination with a decision tree or random forest [24]. On its own, catch22 is fast, but highly inaccurate: see [8, 27]. The Canonical Interval Forest (CIF) is a recent method which adapts the Time Series Forest (TSF) to use catch22 features [27]. CIF is significantly more accurate than either catch22 or TSF.

HIVE-COTE is an ensemble of other methods including BOSS and TSF. Two recent variants of HIVE-COTE, namely HIVE-COTE/TDE (using TDE in place of BOSS) and HIVE-COTE/CIF (using CIF in place of TSF) have been shown to be significantly more accurate than HIVE-COTE, or any other existing method for time series classification [27, 28]. These variants are, in turn, based on an updated 'base' version of HIVE-COTE [2].

While state of the art in terms of accuracy, with the exception of cBOSS these methods are limited by high computational complexity, requiring days or even weeks to train on the datasets in the UCR archive. While more scalable, cBOSS is significantly less accurate than most of the other methods.

2.2 Rocket

Rocket achieves state-of-the-art accuracy, matching the most accurate methods for time series classification (with the exception of the most recent variants of HIVE-COTE), but is considerably faster and more scalable than other methods of comparable accuracy [8].

Rocket transforms input time series using random convolutional kernels, and uses the transformed features to train a linear classifier. Each input time series is convolved with 10,000 random convolutional kernels. Rocket applies global max pooling and PPV (for 'proportion of positive values') pooling to the resulting convolution output to produce two features per kernel per input time series, for a total of 20,000 features per input time series. The transformed features are then used to train a linear classifier: a ridge regression classifier, or logistic regression trained using stochastic gradient descent (for larger datasets).

The kernels are random in terms of their length, weights, bias, dilation, and padding: see Section 3.1. The two most important aspects of Rocket in terms of achieving state-of-the-art accuracy are the use of dilation, sampled on an exponential scale, and the use of PPV. Rocket forms the basis for MiniRocket. The differences between Rocket and MiniRocket are detailed in Section 3.

2.3 Other Methods

The use of a small, fixed set of kernels differentiates MiniRocket from both Rocket, which uses random kernels, and convolutional neural networks such as InceptionTime, which use learned kernels. It also differentiates MiniRocket from other methods with at least superficial similarities to Rocket, such as random shapelet methods as in [18], and other random methods such as those based on [32].

In using kernels with weights constrained to two values (see Section 3.1.2), there are obvious similarities with binary and quantised convolutional neural networks [14, 33]. MiniRocket makes use of at least two advantages of binary/quantised kernels, namely, the ability to perform the convolution operation via addition, as well as efficiencies arising from the relatively small number of possible binary kernels of a given size, e.g., [14, 17, 33]. However, while the kernels used in MiniRocket are binary in the sense of having only two values, these values are *not* 0 and 1 (or −1 and 1). In fact, the actual values of the weights are not important: see Section 3.1.2.

MiniRocket does not use bitwise operations, and the input and convolution output are used at full precision.

The optimisations used in MiniRocket are similar in motivation to several optimisations developed for convolutional neural networks, i.e., broadly speaking, to reduce the number of operations (especially multiplications) required to perform the convolution operation, e.g., [6, 21, 23, 26]. In precomputing the product of the kernel weights and the input, and using those precomputed values to construct the convolution output (see Sections 3.2.3 and 3.2.4), the optimisations used in MiniRocket bear some resemblance to highly simplified versions of shift-based methods [37], where conventional convolutional kernels are replaced by a combination of 1×1 convolutions and spatial shifts in the input, and lookup-based methods [1], where the convolution operation is performed via linear combinations of the precomputed convolution output for a small 'dictionary' of kernels.

However, the optimisations used in MiniRocket are much simpler than these methods. MiniRocket uses a fixed set of kernels, and uses the convolution output for these kernels directly, rather than through a learned linear combination, c.f., e.g., [1, 17]. The optimisations arise as a natural result of using this fixed set of kernels, rather than being general-purpose optimisations.

Several things further distinguish MiniRocket (and Rocket) from most approaches involving convolutional neural networks. The features produced by the transform are all independent of each other (there is no hidden layer). Neither the convolution output nor the pooled features are transformed through, e.g., a sigmoid function or rectified linear unit (ReLU). As such, the classifier learns a direct linear function of the features produced by the transform. MiniRocket is also distinguished by its use of dilation (similar to using many different dilations in a single convolutional layer, with dilations taking any integer value not just powers of two), and PPV.

3 METHOD

MiniRocket involves making certain key changes in order to remove almost all randomness from Rocket (Section 3.1), and exploiting these changes in order to dramatically speed up the transform (Section 3.2). In tuning kernel length, weights, bias, etc., we have restricted ourselves to the same 40 'development' datasets as used in [8], with the same aim of avoiding overfitting the entire UCR archive. (Note, however, that it is not necessarily the aim of MiniRocket to maximise accuracy *per se*, but rather to balance accuracy with parameter choices which remove randomness and are conducive to optimising the transform.) The procedures for setting the parameter values and performing the transform are set out in `fit` and `transform` in Appendix B.

As for Rocket, we implement MiniRocket in Python, compiled via Numba [20]. We use a ridge regression classifier from scikit-learn [31], and logistic regression implemented using PyTorch [30]. Our code is available at: https://github.com/angus924/minirocket.

3.1 Removing Randomness

Rocket uses kernels with lengths selected randomly from $\{7, 9, 11\}$, weights drawn from $\mathcal{N}(0, 1)$, bias terms drawn from $\mathcal{U}(-1, 1)$, random dilations, and random paddings. Two features, PPV and max, are computed per kernel, for a total of 20,000 features. MiniRocket is characterised by a number of key changes to the kernels in terms of length, weights, bias, dilation, and padding, as well as resulting changes to the features, as summarised in Table 1.

Table 1: Summary of changes from Rocket to MiniRocket.

	Rocket	MiniRocket
length	$\{7, 9, 11\}$	9
weights	$\mathcal{N}(0, 1)$	$\{-1, 2\}$
bias	$\mathcal{U}(-1, 1)$	from convolution output
dilation	random	fixed (rel. to input length)
padding	random	fixed
features	PPV + max	PPV
num. features	20K	10K

3.1.1 Length. MiniRocket uses kernels of length 9, with weights restricted to two values, building on the observation in [8] that weights drawn from $\{-1, 0, 1\}$ produce similar accuracy to weights drawn from $\mathcal{N}(0, 1)$.

In order to maximise computational efficiency, the set of kernels should be as small as possible: see Section 3.2.2. The set of possible two-valued kernels grows exponentially with length. There are $2^3 = 8$ possible kernels of length 3, but $2^{15} = 32{,}768$ possible kernels of length 15. With more than two values, the set of possible kernels grows even faster with length. For example, there are $3^{15} \approx 14$ million possible three-valued kernels of length 15.

There are $2^9 = 512$ possible two-valued kernels of length 9. MiniRocket uses a subset of 84 of these kernels, a subset which balances accuracy with the computational advantages of using a small number of kernels: see Section 4.3.1. (A length of 9 is also consistent with the average length used in Rocket.)

3.1.2 Weights. Kernels with weights restricted to two values, α and β, can be characterised in terms of the number of weights with the value β (or, equivalently, the number of weights with the value α). In this sense, the full set of two-valued kernels of length 9 includes the subset of kernels with 1 value of β (e.g., $[\alpha, \alpha, \alpha, \alpha, \alpha, \alpha, \alpha, \alpha, \beta]$), the subset of kernels with 2 values of β (e.g., $[\alpha, \alpha, \alpha, \alpha, \alpha, \alpha, \alpha, \beta, \beta]$), and so on. MiniRocket uses the subset kernels with 3 values of β:

$$[\alpha, \alpha, \alpha, \alpha, \alpha, \alpha, \beta, \beta, \beta]$$
$$[\alpha, \alpha, \alpha, \alpha, \alpha, \beta, \alpha, \beta, \beta]$$
$$[\alpha, \alpha, \alpha, \alpha, \beta, \alpha, \alpha, \beta, \beta]$$
$$\ldots$$

For MiniRocket, we set $\alpha = -1$ and $\beta = 2$. However, the choice of α and β is arbitrary, in the sense that the scale of these values is unimportant. For an input time series, X, kernel, W, and bias, b, PPV is given by $\text{PPV}(X * W - b) = \frac{1}{n} \sum [X * W - b > 0]$ or, equivalently, $\text{PPV}(X * W) = \frac{1}{n} \sum [X * W > b]$, where '$*$' denotes convolution, and $[X \in a]$ denotes the indicator function. As such, computing PPV is essentially equivalent to computing the empirical cumulative distribution function. Accordingly, the scale of the weights is unimportant, because bias values are drawn from the convolution output, $X * W$ (see Section 3.1.3), and so by definition match the scale of the weights and the scale of the input. (Hence, in contrast to Rocket, it is not necessary to normalise the input.)

It is only important that the sum of the weights should be zero or, equivalently, that $\beta = -2\alpha$. Otherwise, the values of α and β are not important. This constraint—that the weights sum to zero—ensures that the kernels are only sensitive to the relative magnitude of the values in the input, i.e., that the convolution output is invariant to the addition or subtraction of any constant value to the input, i.e., $X * W = (X \pm c) * W$.

As PPV is bounded between 0 and 1, in computing PPV for a given kernel, W, we get an equivalent feature for the inverted kernel, $-W$, 'for free': see Section 3.2.1. Accordingly, there is no need to use both the set of kernels with weights $\alpha = -1$ and $\beta = 2$, and the corresponding inverted set of kernels with weights $\alpha = 1$ and $\beta = -2$, as we get these inverted kernels 'for free'.

The set of 84 kernels of length 9 with three weights with the value $\beta = 2$, and six weights with the value $\alpha = -1$, has the desirable properties of being a relatively small, fixed set of kernels—conducive to the optimisations pursued in Section 3.2—and producing high classification accuracy. However, we stress that there is not necessarily anything 'special' about this set of kernels. Other subsets of kernels of length 9, and kernels of other lengths, produce similar accuracy: see Section 4.3.1. This is in addition to the observations in [8], i.e., that kernels (of various lengths) with weights drawn from $\mathcal{N}(0, 1)$, or from $\{-1, 0, 1\}$, are also effective.

3.1.3 Bias. Bias values are drawn from the convolution output, and are used to compute PPV as set out above in Section 3.1.2. By default, for a given kernel/dilation combination, bias values are drawn from the quantiles of the convolution output for a single, randomly-selected training example. For a given kernel, W, and dilation, d, we compute the convolution output for a randomly-selected training example, X, i.e., $W_d * X$. We take, e.g., the $[0.25, 0.5, 0.75]$ quantiles from $W_d * X$ as bias values, to be used in computing PPV. We use a low-discrepancy sequence to assign quantiles to different kernel/dilation combinations [35].

The selection of training examples for the purpose of sampling bias values is the only stochastic element of MiniRocket. Further, while the choice of training example is random, in drawing bias values from the convolution output, we are selecting values produced by an otherwise entirely deterministic procedure. This is why we characterise MiniRocket as 'minimally random'.

For the deterministic variant of MiniRocket, bias values are drawn from the convolution output for the entire training set, rather than a single, randomly-selected training example. This is the only substantive difference between the default and deterministic variants, and the difference in accuracy between the two variants is negligible: see Section 4.1.

The advantage of using the entire training set is an entirely deterministic transform, for applications where this is desirable. However, this comes at additional computational cost, which is unlikely to be practical for larger datasets. Crucially, however, it demonstrates that the accuracy of Rocket is achievable using an entirely deterministic transform. In practice, using a single, randomly-selected training example has little impact in terms of accuracy.

A variant of Rocket using the same method for sampling bias values as MiniRocket is slightly more accurate than default Rocket but, overall, the difference is relatively minor: see Section 4.1.

3.1.4 Dilation. Dilation is used to 'spread' a kernel over the input. For dilation, d, a given kernel is convolved with every d^{th} element of the input [8, 38]. Each kernel is assigned the same fixed set of dilations, adjusted to the length of the input time series. We specify dilations in the range $D = \{\lfloor 2^0 \rfloor, ..., \lfloor 2^{\max} \rfloor\}$, where the exponents are uniformly spaced between 0 and $\max = \log_2(l_{\text{input}} - 1)/(l_{\text{kernel}} - 1)$, where l_{input} is input length and l_{kernel} is kernel length (i.e., 9), such that the maximum effective length of a kernel, including dilation, is the length of the input time series. The count of each unique integer dilation value in D determines the number of features to be computed per dilation (scaled according to the total number of features), ensuring that, as in Rocket, exponentially more features are computed for smaller dilations.

As time series length increases, the number of possible dilation values increases. This means that, for a fixed number of features, the number of features computed per dilation decreases (unless constrained in some way), making the transform less efficient: see Section 3.2.2. Hence, by default, we limit the maximum number of dilations per kernel to 32. While technically an additional hyperparameter, this has little effect on accuracy (see Section 4.3.5), and is intended to be kept at its default value.

3.1.5 Padding. Padding is alternated for each kernel/dilation combination such that, overall, half the kernel/dilation combinations use padding, and half do not. As for Rocket, MiniRocket uses standard zero padding. In effect, zeros are added to the start and end of each input time series such that the convolution operation begins with the 'middle' element of the kernel centered on the first element of the time series, and ends with the 'middle' element of the kernel centered on the last element of the time series [13].

3.1.6 Features. Given the other changes, there is no longer any benefit in terms of accuracy in using global max pooling in addition to PPV: see Section 4.3.3. Accordingly, MiniRocket 'drops' global max pooling and uses only PPV.

We do not replace global max pooling with additional PPV features. As for Rocket, the number of features represents a tradeoff between accuracy and computational expense. MiniRocket with 10,000 features already matches Rocket in terms of accuracy, and there is little or no benefit in terms of accuracy to increasing the number of features beyond 10,000: see Section 4.3.4.

Accordingly, by default, MiniRocket uses 10,000 features (or, more precisely, the nearest multiple of 84—the number of kernels—less than 10,000, i.e., 9,996). While technically a hyperparameter, this is intended to be kept at its default value.

3.2 Optimising the Transform

MiniRocket takes advantage of the properties of the small, fixed set of two-valued kernels, and of PPV, to significantly speed up the transform through four key optimisations:

(1) computing PPV for W and $-W$ at the same time;
(2) reusing the convolution output to compute multiple features;
(3) avoiding multiplications in the convolution operation; and
(4) for each dilation, computing all kernels (almost) 'at once'.

3.2.1 Computing PPV for W and $-W$ at the Same Time. For $C = X * W - b$, PPV is given by $\text{PPV}(C) = \frac{1}{n} \sum [c > 0]$. PPV is bounded between 0 and 1. By definition, the proportion of negative values (or

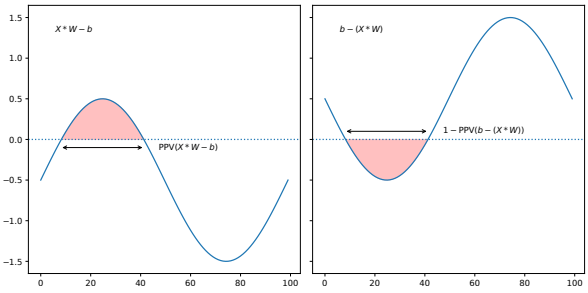

Figure 3: Illustration of $\text{PPV}(X * W - b) = 1 - \text{PPV}(b - (X * W))$**.**

PNV) is the complement of PPV, i.e., $1 - \text{PPV}(X * W - b) = \text{PNV}(X * W - b)$. That is, by computing PPV, we also implicitly compute PNV and vice versa. In this sense, PPV and PNV are equivalent.

Further, the convolution operation is associative, such that $X * -W = -(X * W)$. Accordingly, by computing PPV for a given kernel, W, we unavoidably also compute an equivalent feature (i.e., PNV) for $-W$, that is, $\text{PPV}(X * W - b) = 1 - \text{PPV}(b - (X * W))$. This relationship is illustrated in Figure 3. This means that, for the purposes of PPV, it is unnecessary to compute both $X * W$ and $X * -W$. In fact, it would be redundant to do so.

This means that, in practice, for a given set of kernels where each kernel, W, is matched by a corresponding inverted kernel, $-W$, we only need to perform the convolution operation for W, i.e., for half of the kernels. We get $-W$ 'for free'. Accordingly, as set out in Section 3.1.2, MiniRocket only uses a set of kernels with weights $\alpha = -1$ and $\beta = 2$, as it is unnecessary to also use the corresponding set of inverted kernels with weights $\alpha = 1$ and $\beta = -2$.

3.2.2 Reusing the Convolution Output. For MiniRocket, the same kernel/dilation combination is used to compute multiple features, at least for smaller dilations (exponentially fewer features are computed for larger dilations: see Section 3.1.4).

For a given kernel, W, and dilation, d, we compute $C = X * W_d$ and then reuse the convolution output, C, to compute multiple features, i.e., for multiple different bias values. This has the effect that multiple features are computed with the computational cost of a single convolution operation, plus the much lower cost of computing PPV for each bias value.

3.2.3 Avoiding Multiplications. Restricting the kernel weights to two values allows us to, in effect, 'factor out' the multiplications from the convolution operation, and to perform the convolution operation using only addition.

For input time series $X = [x_0, x_1, ..., x_{n-1}]$, and kernel $W = [w_0, w_1, ..., w_{m-1}]$, with dilation, d, the convolution operation can be formulated as:

$$X * W_d = \sum_{j=0}^{m-1} x_{i-(\lfloor \frac{m}{2} \rfloor \cdot d)+(j \cdot d)} \cdot w_j, \forall i \in \{0, 1, ..., n-1\}.$$

Equivalently, the convolution operation can be thought of as the column sums of a matrix, $\hat{C}$, where each row corresponds to the input time series multiplied by the appropriate kernel weight, and the alignment of the rows corresponds to dilation (values of 0 in $\hat{C}$ represent zero padding), e.g.:

$$\hat{C} = \begin{bmatrix} 0 & 0 & 0 & 0 & w_0x_0 & \cdots \\ 0 & 0 & 0 & w_1x_0 & w_1x_1 & \cdots \\ 0 & 0 & w_2x_0 & w_2x_1 & w_2x_2 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \\ w_{m-1}x_4 & w_{m-1}x_5 & w_{m-1}x_6 & w_{m-1}x_7 & w_{m-1}x_8 & \cdots \end{bmatrix}$$

The result of the convolution operation is given by the column sums of $\hat{C}$, i.e., $C = X * W = \mathbf{1}^\top \hat{C}$, where $\mathbf{1}$ is a vector, $[1, 1, ..., 1]^\top$, of length n.

Where the weights of the kernels are restricted to two values, α and β, we can 'factor out' the multiplications by precomputing $A = \alpha X$ and $B = \beta X$ and then, for a given kernel, e.g., $W = [\alpha, \beta, \alpha, ..., \alpha]$, completing the convolution operation by summation using $A = [a_0, a_1, ..., a_{n-1}]$ and $B = [b_0, b_1, ..., b_{n-1}]$:

$$\hat{C} = \begin{bmatrix} 0 & 0 & 0 & 0 & a_0 & \cdots & a_{n-5} \\ 0 & 0 & 0 & b_0 & b_1 & \cdots & b_{n-4} \\ 0 & 0 & a_0 & a_1 & a_2 & \cdots & b_{n-3} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_4 & a_5 & a_6 & a_7 & a_8 & \cdots & 0 \end{bmatrix}$$

In other words, it is only necessary to compute αX and βX once for each input time series, and then reuse the results to complete the convolution operation for each kernel by addition.

3.2.4 Computing All the Kernels (Almost) 'At Once'. We can take further advantage of using only two values for the kernel weights in order to perform most of the computation required for all 84 kernels 'at once' for each dilation value. More precisely, as MiniRocket uses kernels with six weights of one value, and three weights of another value, we can perform $\frac{6}{9} = \frac{2}{3}$ of the computation for all 84 kernels 'at once' for a given dilation.

This is possible by treating all kernel weights as $\alpha = -1$, precomputing convolution output, C_α, and later adjusting C_α for each kernel. Per Section 3.2.3, C_α can be thought of as the column sums of a matrix with 9 rows, where each row corresponds to $\alpha X = -X$, aligned according to dilation. For example, for a dilation of 1:

$$\hat{C}_\alpha = \begin{bmatrix} 0 & 0 & 0 & 0 & -x_0 & \cdots & -x_{n-5} \\ 0 & 0 & 0 & -x_0 & -x_1 & \cdots & -x_{n-4} \\ 0 & 0 & -x_0 & -x_1 & -x_2 & \cdots & -x_{n-3} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ -x_4 & -x_5 & -x_6 & -x_7 & -x_8 & \cdots & 0 \end{bmatrix}$$

For MiniRocket, the kernel weights are $\alpha = -1$ and $\beta = 2$. Let $\gamma = 3$, noting that $2 = -1 + 3$. As for $\hat{C}_\alpha$, we then form $\hat{C}_\gamma$, where each row corresponds to $\gamma X = 3X$, aligned according to dilation. For each kernel, C_γ is equivalent to the column sums of those rows in $\hat{C}_\gamma$ corresponding to the position of the β weights in the given kernel. For example, for kernel $W = [\beta, \alpha, \beta, \alpha, \beta, \alpha, \alpha, \alpha, \alpha]$:

$$\hat{C}_\gamma^{(W)} = \begin{bmatrix} 0 & 0 & 0 & 0 & 3x_0 & \cdots & 3x_{n-5} \\ 0 & 0 & 3x_0 & 3x_1 & 3x_2 & \cdots & 3x_{n-3} \\ 3x_0 & 3x_1 & 3x_2 & 3x_3 & 3x_4 & \cdots & 3x_{n-1} \end{bmatrix}$$

The final convolution output for a given kernel is then given by $C = C_\alpha + C_\gamma$. In other words, we can reuse C_α, computed once for a given dilation, to compute the convolution output for all 84 kernels

for that dilation. For each kernel, computing C only involves adding C_γ to C_α. In performing the convolution operation in this way, we only have to compute C_γ for each kernel, i.e., $\frac{1}{3}$ of the computation otherwise required.

3.3 Classifiers

Like Rocket, MiniRocket is a transform, producing features which are then used to train a linear classifier. We use the same classifiers as Rocket to learn the mapping from the features to the classes, i.e., a ridge regression classifier or, for larger datasets, logistic regression trained using Adam [19]. As for Rocket, we suggest switching from the ridge regression classifier to logistic regression when there are more training examples than features, i.e., when there are more than approximately 10,000 training examples.

3.4 Complexity

Fundamentally, the scalability of MiniRocket remains the same as for Rocket: linear in the number of kernels/features (k), the number of examples (n), and time series length (l_{input}) or, formally, $O(k \cdot n \cdot l_{\text{input}})$. While MiniRocket uses a smaller number of kernel/dilation combinations, and computes multiple features for each kernel/dilation combination, complexity is still proportional to the number of kernels/features. Similarly, while MiniRocket 'factors out' multiplications from the convolution operation, the number of addition operations is still proportional to the number of kernels and time series length, and while MiniRocket performs the majority of the computation required for all 84 kernels 'at once', the remaining computation is still proportional to the number of kernels/features. However, within this broad class of complexity, the various optimisations pursued in Section 3.2 make MiniRocket significantly faster in practice.

3.5 Memory

Compared to Rocket (which does not store any intermediate values), MiniRocket temporarily stores up to 13 additional vectors, namely, $A = -X$, $G = \gamma X = 3X$ (plus 9 variants of G pre-aligned for the given dilation), C_α, and C: see Sections 3.2.3 and 3.2.4. This is equivalent to storing 13 additional copies of a single input time series (approx. $1{,}000{,}000 \times 4 \times 13 = 52$MB for time series of length 1 million), which should be negligible in almost all cases.

When transforming the training set, the deterministic variant stores the convolution output for a given kernel/dilation combination for the entire training set, which is equivalent to storing one additional copy of the entire training set. This is impractical for larger datasets, which is why it is avoided by default.

4 EXPERIMENTS

We evaluate MiniRocket on the datasets in the UCR archive (Section 4.1), showing that, on average, MiniRocket is marginally more accurate than Rocket, and not significantly less accurate than the most accurate current methods for time series classification. We demonstrate the speed and scalability of MiniRocket in terms of both training set size and time series length (Section 4.2), showing that MiniRocket is up to 75 times faster than Rocket on larger datasets. We also explore the effect of key parameters in relation to kernel length, bias, output features, and dilation (Section 4.3).

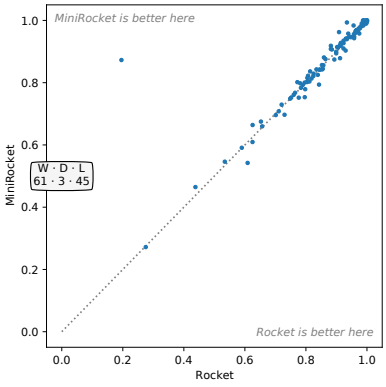

Figure 4: Pairwise accuracy of MiniRocket versus Rocket.

4.1 UCR Archive

We evaluate MiniRocket on the datasets in the UCR archive [7]. We compare MiniRocket against the most accurate current methods for time series classification, namely, HIVE-COTE/TDE (representative of HIVE-COTE and its variants), TS-CHIEF, InceptionTime, and Rocket, as well as TDE, CIF, cBOSS and Proximity Forest.

For consistency and direct comparability with the most recent published results for other state-of-the-art methods [2, 27, 28], we evaluate MiniRocket on 30 resamples of 109 datasets from the archive. We use the same 30 resamples (including the default training/test split) as in [2, 27, 28]. (Full results are available in the accompanying repository.)

Figure 1 on page 1 shows the mean rank of MiniRocket versus the other state-of-the-art methods. Methods for which the pairwise difference in accuracy is not statistically significant, per a Wilcoxon signed-rank test with Holm correction (as a post hoc test to the Friedman test), are connected with a black line [4, 9, 12].

MiniRocket is, on average, marginally more accurate than Rocket, and somewhat less accurate than the most accurate current methods, namely TS-CHIEF and HIVE-COTE/TDE, although the differences in accuracy are not statistically significant. However, as noted in Section 1, the total compute time for MiniRocket on these datasets is a tiny fraction of the total compute time required by the other methods (even Rocket, which is already considerably faster than even the fastest of the other methods).

MiniRocket versus Rocket. Figure 4 shows the pairwise accuracy of MiniRocket versus Rocket for the same 109 datasets. Overall, MiniRocket and Rocket achieve very similar accuracy. MiniRocket is more accurate than Rocket on 61 datasets, and less accurate on 45 datasets, but the differences in accuracy are mostly small. The large difference in accuracy between MiniRocket and Rocket on one dataset, *PigAirwayPressure*, appears to be due to the way the bias values are sampled. We also evaluated a variant of Rocket which uses the same method of sampling bias values as MiniRocket. Overall, this variant is slightly more accurate than default Rocket, but the difference is relatively minor, with a win/draw/loss of 50/6/53 against MiniRocket.

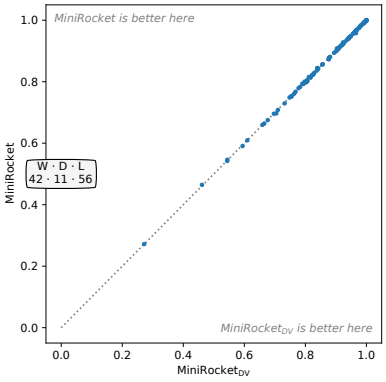

Figure 5: Pairwise accuracy of default MiniRocket versus the deterministic variant.

Table 2: Accuracy and total training time.

	Accuracy		Training Time	
	Rocket	MiniRocket	Rocket	MiniRocket
Fruit	0.9491	0.9568	2h 36m 40s	2m 22s
Insect	0.7796	0.7639	26m 44s	37s
Mosquito	0.8271	0.8165	15h 34m 58s	12m 32s

Deterministic variant. Figure 5 shows the pairwise accuracy of default MiniRocket vs the deterministic variant (or MiniRocket$_{DV}$) for the same 109 datasets. Overall, the deterministic variant produces essentially the same accuracy as the default variant.

4.2 Scalability

4.2.1 Training Set Size. We demonstrate the speed and scalability of MiniRocket in terms of training set size on the three largest datasets in the UCR archive, namely, *MosquitoSound* (139,780 training examples, each of length 3,750), *InsectSound* (25,000 training examples, each of length 600), and *FruitFlies* (17,259 training examples, each of length 5,000). These recent additions are significantly larger than other datasets in the archive.

For this purpose, following [8], we integrate MiniRocket (and Rocket) with logistic regression, trained using Adam. Training details are provided in Appendix A. The experiments were performed on the same cluster as noted in Section 1 and, again, both Rocket and MiniRocket are restricted to a single CPU core.

Figure 6 shows training time vs training set size for MiniRocket and Rocket. Training time includes the transform for both validation and training sets, and classifier training. Table 2 shows test accuracy and total training time (for the full training set).

MiniRocket is slightly more accurate on one of the datasets, and slightly less accurate on two of the datasets. This is consistent with the small differences in accuracy observed on the other datasets in the UCR archive: see Section 4.1. However, MiniRocket is considerably faster than Rocket: 43 times faster on *InsectSound*, 66 times faster on *FruitFlies*, and 75 times faster on *MosquitoSound*.

The accuracy of Rocket and MiniRocket on the *InsectSound* and *MosquitoSound* datasets appears to be broadly comparable to reported results for other methods for these datasets or versions of these datasets [5, 10, 11, 39], although some deep learning approaches are significantly more accurate on *MosquitoSound* [10].

4.2.2 Time Series Length. We demonstrate the scalability of MiniRocket in terms of time series length on the dataset in the UCR archive with the longest time series, *DucksAndGeese* (50 training examples, each of length 236,784). This recent addition has significantly longer time series than other datasets in the archive.

Figure 6 shows training time versus time series length for both Rocket and MiniRocket. Training time includes the transform and classifier training. (With only 50 training examples, we use the ridge regression classifier.)

While both Rocket and MiniRocket are linear in time series length, MiniRocket is considerably faster for a given length. With more training examples, we would expect the difference in training time to be considerably larger. With only 50 training examples, the overhead of sampling bias values (which is unrelated to training set size) constitutes a significant proportion of the total training time for MiniRocket.

4.3 Sensitivity Analysis

We explore the effect of key parameter choices on accuracy:

- kernel length;
- sampling bias from the convolution output versus $\mathcal{U}(-1, 1)$;
- using only PPV versus both PPV and global max pooling;
- the number of features; and
- limiting the maximum number of dilations per kernel.

We perform the analysis using the 40 'development' datasets (default training/test splits). Results are mean results over 10 runs.

4.3.1 Kernels. Figure 7 shows the effect of kernel length on accuracy. For kernels of length 9, a subscript refers to a particular subset of kernels in the sense discussed in Section 3.1.2. (E.g., $9_{\{3\}}$ refers to kernels with three weights of one value, and six weights of another value.) The total number of features is kept constant (to the nearest multiple of the number of kernels less than 10,000: see Section 3.1.6), such that more features are computed per kernel for smaller sets of kernels and vice versa.

Kernels of length 9 are most accurate, but kernels of length 7 or 11 are not significantly less accurate. This is consistent with the findings in [8] in relation to Rocket. The actual differences in accuracy between kernels of different lengths is very small.

Crucially, however, as noted in Section 3.1.2, the $9_{\{3\}}$ subset is nearly as accurate as the full set of kernels of length 9. This is a relatively small subset of kernels, and is particularly well suited to the optimisations pursued in Section 3.2.

4.3.2 Bias. Figure 8 shows the effect in terms of accuracy of sampling bias from the convolution output versus from $\mathcal{U}(-1, 1)$ as in Rocket. MiniRocket is significantly less accurate when sampling bias from $\mathcal{U}(-1, 1)$. The change to sampling bias from the convolution output is critical to matching the accuracy of Rocket.

4.3.3 Features. Figure 9 shows the effect of using only PPV versus both PPV and global max pooling. With the other changes to

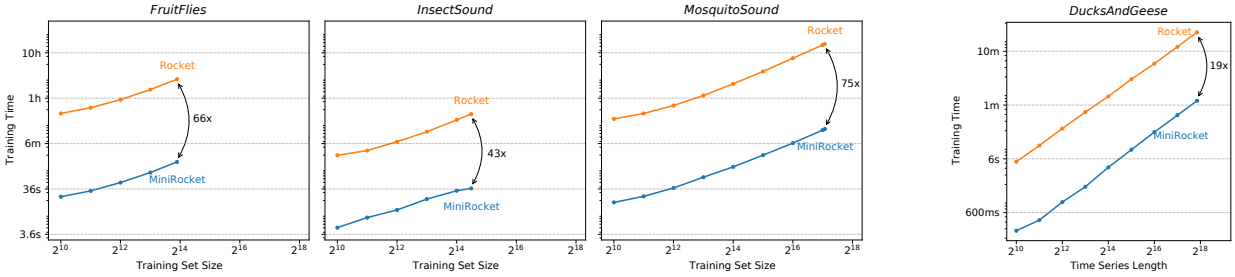

Figure 6: Training time versus (left) training set size and (right) time series length.

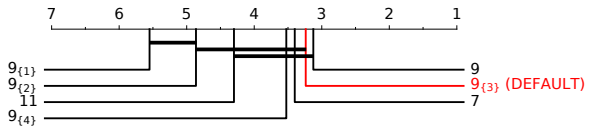

Figure 7: Mean rank for different kernel lengths.

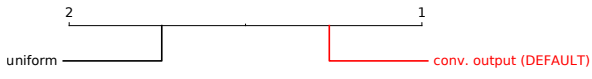

Figure 8: Mean rank for bias sampled from the convolution output versus bias sampled from $\mathcal{U}(-1, 1)$.

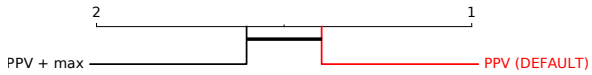

Figure 9: Mean rank for PPV vs PPV and global max pooling.

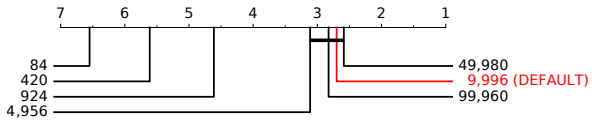

Figure 10: Mean rank for different numbers of features.

MiniRocket—in particular, with the change to sampling bias from the convolution output—there is no advantage to using global max pooling in addition to PPV. In fact, using global max pooling in addition to PPV is less accurate than just using PPV, although the difference is not statistically significant.

4.3.4 Number of Features. Figure 10 shows the effect of different numbers of features between 84 and 99,960 (the nearest multiple of 84 less than 100, 500, 1,000, ...). Increasing the number of features noticeably increases accuracy up to approximately 10,000 features. There is little or no benefit to increasing the number of features beyond 10,000, at least for shorter time series, because there is little benefit in computing PPV for many more than l_{input} bias values for time series of length l_{input} (more and more features will be the same). For 49,980 and 99,960 features, we have endeavoured to avoid this limitation as much as possible by setting the maximum number of dilations per kernel to 119 (see Section 3.1.4) and, where necessary, sampling bias values from multiple training examples.

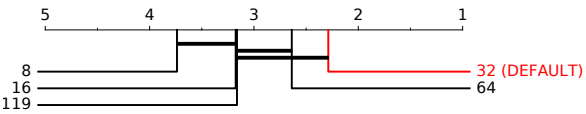

Figure 11: Mean rank of different values for the maximum number of dilations per kernel.

4.3.5 Dilation. Figure 11 shows the effect in terms of accuracy of different values for the maximum number of dilations per kernel. The total number of features is kept constant, such that more features are computed per dilation for a smaller number of maximum dilations per kernel and vice versa: see Section 3.1.4. There is little difference in accuracy between values of 16 and 119 (119 being the largest possible number of dilations per kernel for the default number of features, i.e., $\lfloor 10{,}000/84 \rfloor = 119$). A value of 32 balances accuracy with the computational advantage of limiting the number of dilations per kernel, as discussed in Section 3.1.4.

5 CONCLUSION

We reformulate Rocket into a new method, MiniRocket, making it up to 75 times faster on larger datasets. MiniRocket shows that it is possible to achieve essentially the same accuracy as Rocket using a mostly-deterministic and much faster procedure.

MiniRocket represents a significant advance in accuracy relative to computational cost. MiniRocket is much faster than any other method of comparable accuracy (including Rocket), and far more accurate than any other method of even roughly-similar computational expense. Accordingly, we suggest that MiniRocket should be considered and used as the default variant of Rocket. We provide a naïve facility for applying MiniRocket to multivariate time series (available through the accompanying repository). In future work, we propose to investigate more sophisticated approaches to multivariate time series, to explore the integration of MiniRocket with nonlinear classifiers, and the use of MiniRocket beyond time series data.

ACKNOWLEDGMENTS

This material is based on work supported by an Australian Government Research Training Program Scholarship, and the Australian Research Council under award 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 mean ranks were produced using code from [15].

REFERENCES

[1] Hessam Bagherinezhad, Mohammad Rastegari, and Ali Farhadi. 2017. LCNN: Lookup-based Convolutional Neural Network. In *2017 IEEE Conference on Computer Vision and Pattern Recognition*. 860–869.

[2] Anthony Bagnall, Michael Flynn, James Large, Jason Lines, and Matthew Middlehurst. 2020. On the Usage and Performance of the Hierarchical Vote Collective of Transformation-Based Ensembles Version 1.0 (HIVE-COTE v1.0). In *ECML PKDD Workshop on Advanced Analytics and Learning on Temporal Data*.

[3] Anthony Bagnall, Jason Lines, Aaron Bostrom, James Large, and Eamonn Keogh. 2017. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery* 31, 3 (2017), 606–660.

[4] Alessio Benavoli, Giorgio Corani, and Francesca Mangili. 2016. Should We Really Use Post-Hoc Tests Based on Mean-Ranks? *Journal of Machine Learning Research* 17, 5 (2016), 1–10.

[5] Yanping Chen, Adena Why, Gustavo Batista, Agenor Mafra-Neto, and Eamonn Keogh. 2014. Flying Insect Classification with Inexpensive Sensors. *Journal of Insect Behavior* 27, 5 (2014), 657–677.

[6] François Chollet. 2017. Xception: Deep Learning with Depthwise Separable Convolutions. In *2017 IEEE Conference on Computer Vision and Pattern Recognition*. 1800–1807.

[7] Hoang An Dau, Anthony Bagnall, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, and Eamonn Keogh. 2019. The UCR Time Series Archive. *IEEE/CAA Journal of Automatica Sinica* 6, 6 (2019), 1293–1305.

[8] Angus Dempster, François Petitjean, and Geoffrey I Webb. 2020. ROCKET: Exceptionally fast and accurate time classification using random convolutional kernels. *Data Mining and Knowledge Discovery* 34, 5 (2020), 1454–1495.

[9] Janez Demšar. 2006. Statistical Comparisons of Classifiers over Multiple Data Sets. *Journal of Machine Learning Research* 7 (2006), 1–30.

[10] Eleftherios Fanioudakis, Matthias Geismar, and Ilyas Potamitis. 2018. Mosquito wingbeat analysis and classification using deep learning. In *European Signal Processing Conference*. 2424–2428.

[11] Michael Flynn and Anthony Bagnall. 2019. Classifying Flies Based on Reconstructed Audio Signals. In *Intelligent Data Engineering and Automated Learning*, Hujun Yin, David Camacho, Peter Tino, Antonio J. Tallón-Ballesteros, Ronaldo Menezes, and Richard Allmendinger (Eds.). Springer, Cham, 249–258.

[12] Salvador García and Francisco Herrera. 2008. An Extension on "Statistical Comparisons of Classifiers over Multiple Data Sets" for All Pairwise Comparisons. *Journal of Machine Learning Research* 9 (2008), 2677–2694.

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press, Cambridge, MA.

[14] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2018. Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations. *Journal of Machine Learning Research* 18, 187 (2018), 1–30.

[15] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. 2019. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery* 33, 4 (2019), 917–963.

[16] Hassan Ismail Fawaz, Benjamin Lucas, Germain Forestier, Charlotte Pelletier, Daniel F. Schmidt, Jonathan Weber, Geoffrey I. Webb, Lhassane Idoumghar, Pierre-Alain Muller, and François Petitjean. 2020. InceptionTime: Finding AlexNet for Time Series Classification. *Data Mining and Knowledge Discovery* 34, 6 (2020), 1936–1962.

[17] Felix Juefei-Xu, Vishnu Naresh Boddeti, and Marios Savvides. 2017. Local Binary Convolutional Neural Networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition*. 4284–4293.

[18] Isak Karlsson, Panagiotis Papapetrou, and Henrik Boström. 2016. Generalized random shapelet forests. *Data Mining and Knowledge Discovery* 30, 5 (2016), 1053–1085.

[19] Diederik P. Kingma and Jimmy Lei Ba. 2015. Adam: A Method for Stochastic Optimization. In *Third International Conference on Learning Representations*. arXiv:1412.6980.

[20] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. 2015. Numba: A LLVM-based Python JIT Compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*. 1–6.

[21] Andrew Lavin and Scott Gray. 2016. Fast Algorithms for Convolutional Neural Networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition*. 4013–4021.

[22] Jason Lines, Sarah Taylor, and Anthony Bagnall. 2018. Time Series Classification with HIVE-COTE: The Hierarchical Vote Collective of Transformation-Based Ensembles. *ACM Transactions on Knowledge Discovery from Data* 12, 5 (2018), 52:1–52:35.

[23] Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall Tappen, and Marianna Penksy. 2015. Sparse Convolutional Neural Networks. In *2015 IEEE Conference on Computer Vision and Pattern Recognition*. 806–814.

[24] Carl H. Lubba, Sarab S. Sethi, Philip Knaute, Simon R. Schultz, Ben D. Fulcher, and Nick S. Jones. 2019. catch22: CAnonical Time-series CHaracteristics. *Data Mining and Knowledge Discovery* 33, 6 (2019), 1821–1852.

[25] Benjamin Lucas, Ahmed Shifaz, Charlotte Pelletier, Lachlan O'Neill, Nayyar Zaidi, Bart Goethals, François Petitjean, and Geoffrey I. Webb. 2019. Proximity Forest: an effective and scalable distance-based classifier for time series. *Data Mining and Knowledge Discovery* 33, 3 (2019), 607–635.

[26] Sachin Mehta, Mohammad Rastegari, Anat Caspi, Linda Shapiro, and Hannaneh Hajishirzi. 2018. ESPNet: Efficient Spatial Pyramid of Dilated Convolutions for Semantic Segmentation. In *European Conference on Computer Vision*, Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss (Eds.). Springer, Cham, 561–580.

[27] Matthew Middlehurst, James Large, and Anthony Bagnall. 2020b. The Canonical Interval Forest (CIF) Classifier for Time Series Classification. In *IEEE International Conference on Data Mining*.

[28] Matthew Middlehurst, James Large, Gavin Cawley, and Anthony Bagnall. 2020a. The Temporal Dictionary Ensemble (TDE) Classifier for Time Series Classification. In *ECML PKDD*.

[29] Matthew Middlehurst, William Vickers, and Anthony Bagnall. 2019. Scalable Dictionary Classifiers for Time Series Classification. In *Intelligent Data Engineering and Automated Learning*, Hujun Yin, David Camacho, Peter Tino, Antonio J. Tallón-Ballesteros, Ronaldo Menezes, and Richard Allmendinger (Eds.). Springer, Cham, 11–19.

[30] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, et al. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett (Eds.). 8024–8035.

[31] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, et al. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[32] Ali Rahimi and Benjamin Recht. 2008. Random Features for Large-Scale Kernel Machines. In *Advances in Neural Information Processing Systems 20*, J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis (Eds.). 1177–1184.

[33] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. In *European Conference on Computer Vision*, Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling (Eds.). Springer, Cham, 525–542.

[34] Patrick Schäfer. 2015. The BOSS is concerned with time series classification in the presence of noise. *Data Mining and Knowledge Discovery* 29, 6 (2015), 1505–1530.

[35] Colas Schretter, Zhijian He, Mathieu Gerber, Nicolas Chopin, and Harald Niederreiter. 2016. Van der Corput and Golden Ratio Sequences Along the Hilbert Space-Filling Curve. In *Eleventh International Conference on Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing*, Ronald Cools and Dirk Nuyens (Eds.). Springer, Cham, 531–544.

[36] Ahmed Shifaz, Charlotte Pelletier, François Petitjean, and Geoffrey I. Webb. 2020. TS-CHIEF: A Scalable and Accurate Forest Algorithm for Time Series Classification. *Data Mining and Knowledge Discovery* (2020).

[37] Bichen Wu, Alvin Wan, Xiangyu Yue, Peter Jin, Sicheng Zhao, Noah Golmant, Amir Gholaminejad, Joseph Gonzalez, and Kurt Keutzer. 2018. Shift: A Zero FLOP, Zero Parameter Alternative to Spatial Convolutions. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 9127–9135.

[38] Fisher Yu and Vladlen Koltun. 2016. Multi-Scale Context Aggregation by Dilated Convolutions. In *Fourth International Conference on Learning Representations*. arXiv:1511.07122.

[39] Chongsheng Zhang, Pengyou Wang, Hui Guo, Gaojuan Fan, Ke Chen, and Joni-Kristian Kämäräinen. 2017. Turning wingbeat sounds into spectrum images for acoustic insect classification. *Electronics Letters* 53, 25 (2017), 1674–1676.

A LOGISTIC REGRESSION TRAINING

For each dataset, we shuffle the training set once, and train on increasingly large subsets of the shuffled training data. The transform is performed in batches (of 2^{12} training examples), which are further divided into minibatches for training. We use the same hyperparameters for all datasets: a validation set of 2,048 examples, a minibatch size of 256, an initial learning rate 10^{-4}, the learning rate is halved if validation loss does not improve after 50 updates, and training is stopped if validation loss does not improve after 100 updates. The same hyperparameters can be reused for any dataset, because the classifier 'always sees the same thing', i.e., blocks of transformed features. Additionally, we cache the transformed features, in order to avoid unnecessarily repeating the transform when training for multiple epochs.

B ALGORITHM

Function transform($\boldsymbol{X}, D, N, B$)

input : $\boldsymbol{X}$: time series
D : dilations
N : num features per dilation
B : biases

output: $\boldsymbol{F}$: features

begin
  // indices of β weights in kernels
  $\boldsymbol{I} \leftarrow [[0, 1, 2], [0, 1, 3], \ldots, [6, 7, 8]]$
  for $i \in [0, 1, \ldots, \text{size}(\boldsymbol{X}) - 1]$ **do**
    $a \leftarrow 0$
    $X \leftarrow \boldsymbol{X}[i]$
    $A, G \leftarrow -X, 3X$ // §3.2.3
    for $j \in [0, 1, \ldots, |D| - 1]$ **do**
      $\hat{p}_0 \leftarrow j \bmod 2$
      $p \leftarrow 4 \cdot D[j]$
      precompute $C_\alpha, \hat{\boldsymbol{C}}_\gamma$ for $D[j]$ per §3.2.4
      for $k \in [0, 1, \ldots, 83]$ **do**
        $b \leftarrow a + N[j]$
        $C_\gamma \leftarrow \hat{\boldsymbol{C}}_\gamma[\boldsymbol{I}[k]]$ // §3.2.4
        $C \leftarrow C_\alpha + C_\gamma$
        $\hat{p}_1 \leftarrow \hat{p}_0 + k \bmod 2$ // §3.1.5
        if $\hat{p}_1$ **then** $C \leftarrow C[p{:}|C| - p]$
        $\boldsymbol{F}[i, a{:}b] \leftarrow \text{PPV}(C, B[a{:}b])$
        $a \leftarrow b$
      end
    end
  end
  return $\boldsymbol{F}$
end

Function fit($\boldsymbol{X}, n, m$)

input : $\boldsymbol{X}$: time series
n : num features
m : max dilations per kernel

output: D : dilations
N : num features per dilation
B : biases

begin
  $\max \leftarrow \log_2(\text{length}(\boldsymbol{X}) - 1)/8$ // §3.1.4
  $\hat{D} \leftarrow [\lfloor 2^0 \rfloor, \lfloor 2^{\max/m} \rfloor, \lfloor 2^{2 \cdot \max/m} \rfloor, \ldots, \lfloor 2^{m \cdot \max/m} \rfloor]$
  $D \leftarrow$ unique elements in $\hat{D}$
  $\hat{N} \leftarrow$ count of each unique element in $\hat{D}$
  $N \leftarrow \lfloor \hat{N} \cdot n/84/m \rfloor$ // scale to n, s.t. $\text{sum}(N) \approx n$
  $r \leftarrow n - \text{sum}(N)$ // apportion remainder
  if $r > 0$ **then** $N[:r] \leftarrow N[:r] + 1$
  $Q \leftarrow i \cdot \frac{1+\sqrt{5}}{2} \bmod 1, \forall i \in [1, 2, \ldots, n]$ // §3.1.3
  $B \leftarrow \text{sample}(\boldsymbol{X}, D, N, Q)$
  return D, N, B
end

Function sample($\boldsymbol{X}, D, N, Q$)

input : $\boldsymbol{X}$: time series
D : dilations
N : num features per dilation
Q : quantiles

output: B : biases

begin
  // indices of β weights in kernels
  $\boldsymbol{I} \leftarrow [[0, 1, 2], [0, 1, 3], \ldots, [6, 7, 8]]$
  $a \leftarrow 0$
  for $j \in [0, 1, \ldots, |D| - 1]$ **do**
    for $k \in [0, 1, \ldots, 83]$ **do**
      $b \leftarrow a + N[j]$
      $X \leftarrow \text{random}(\boldsymbol{X})$
      $A, G \leftarrow -X, 3X$ // §3.2.3
      compute $C_\alpha, \hat{\boldsymbol{C}}_\gamma$ for $D[j]$ per §3.2.4
      $C_\gamma \leftarrow \hat{\boldsymbol{C}}_\gamma[\boldsymbol{I}[k]]$
      $C \leftarrow C_\alpha + C_\gamma$
      $B[a{:}b] \leftarrow \text{quantiles}(C, Q[a{:}b])$ // §3.1.3
      $a \leftarrow b$
    end
  end
  return B
end