Data Mining and Knowledge Discovery (2023) 37:1779–1805
https://doi.org/10.1007/s10618-023-00939-3

Hydra: competing convolutional kernels for fast and accurate time series classification

Angus Dempster[1] · **Daniel F. Schmidt**[1] · **Geoffrey I. Webb**[1]

Received: 25 March 2022 / Accepted: 24 April 2023 / Published online: 16 May 2023

Abstract
We demonstrate a simple connection between dictionary methods for time series classification, which involve extracting and counting symbolic patterns in time series, and methods based on transforming input time series using convolutional kernels, namely Rocket and its variants. We show that by adjusting a single hyperparameter it is possible to move by degrees between models resembling dictionary methods and models resembling Rocket. We present Hydra, a simple, fast, and accurate dictionary method for time series classification using competing convolutional kernels, combining key aspects of both Rocket and conventional dictionary methods. Hydra is faster and more accurate than the most accurate existing dictionary methods, achieving similar accuracy to several of the most accurate current methods for time series classification. Hydra can also be combined with Rocket and its variants to significantly improve the accuracy of these methods.

Keywords Time series classification · Dictionary · Random · Convolution · Rocket

1 Introduction

Dictionary methods and Rocket (Dempster et al. 2020) represent two seemingly quite different approaches to time series classification. Dictionary methods involve extracting and then counting symbolic patterns in time series. Rocket and its variants transform time series using convolutional kernels, and could be seen as having more in common with convolutional neural networks.

We demonstrate a simple connection between dictionary methods and Rocket. We show that by adjusting a single hyperparameter it is possible to move by degrees between models more closely resembling dictionary methods and models more

Responsible editor: Myra Spiliopoulou.

✉ Angus Dempster
angus.dempster1@monash.edu

[1] Monash University, Melbourne, Australia

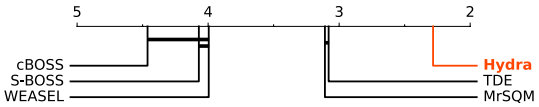

Fig. 1 Mean rank of Hydra in terms of accuracy versus several prominent dictionary methods over 30 resamples of 106 datasets from the UCR archive. Lower rank indicates higher accuracy. Hydra is more accurate than the most accurate existing dictionary methods

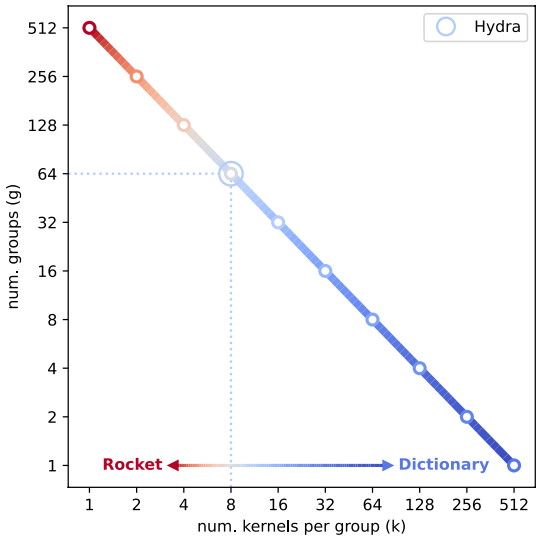

Fig. 2 The number of groups (g) or, equivalently, the number of kernels per group (k), controls the resemblance of Hydra to dictionary methods or Rocket. By default, Hydra uses 64 groups with 8 kernels per group

closely resembling Rocket (Figs. 1, 2). This provides the basis for performing fast and accurate dictionary-like classification using convolutional kernels. We present Hydra (for **HY**brid **D**ictionary–**R**ocket **A**rchitecture), a simple, fast, and accurate dictionary method for time series classification using competing convolutional kernels, incorporating aspects of both Rocket and conventional dictionary methods.

Hydra incorporates two key aspects of dictionary methods: (1) forming groups (dictionaries) of patterns which approximate the input time series; and (2) using the counts of these patterns to perform classification. However, unlike typical dictionary methods, Hydra uses random patterns represented by random convolutional kernels.

Like Rocket, Hydra transforms input time series using random convolutional kernels. However, unlike Rocket: (1) the kernels are arranged into groups; and (2) Hydra counts the kernels in each group representing the closest match with the input at each timepoint, i.e., treating each kernel as a pattern in a dictionary, and forcing the kernels in each group to 'compete' in order to be counted at each timepoint: see Fig. 3.

While superficially simple, the organisation of the kernels into groups is what allows Hydra to perform dictionary-like classification using random patterns with high accuracy. The number of groups (g) or, equivalently, the number of kernels

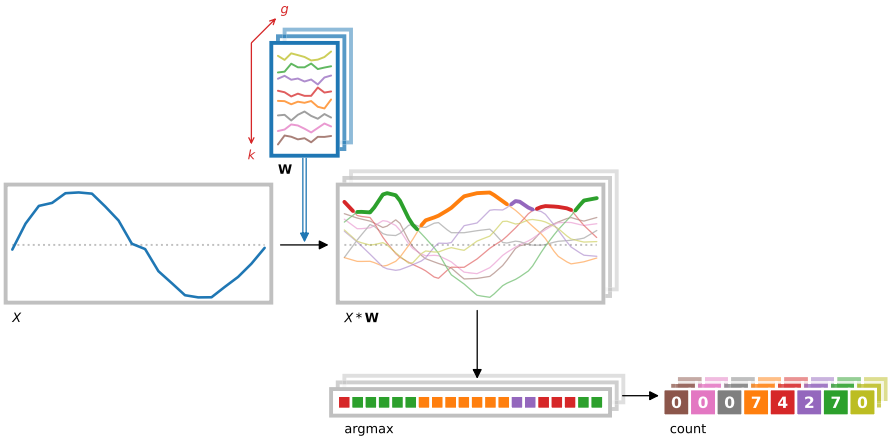

Fig. 3 Hydra convolves each input time series with a set of random convolutional kernels, organised into g groups with k kernels per group, and at each timepoint counts the kernels representing the closest match with the input time series for each group

per group (k),[1] controls the extent to which Hydra more closely resembles dictionary methods or more closely resembles Rocket, and has a decisive influence on the accuracy of the method: see Fig. 2. With multiple kernels per group, Hydra more closely resembles conventional dictionary methods. As the number of kernels per group approaches one ($k \to 1$), Hydra more closely resembles Rocket. In fact, where $k = 1$ ($g \gg 1$), Hydra essentially *is* Rocket, with some qualifications. However, the most Rocket-like variant is *not* the most accurate variant of Hydra.

The use of random patterns in the form of random convolutional kernels has two key advantages: (1) it is computationally efficient; and (2) it produces high classification accuracy. Figure 1 shows the mean rank of Hydra versus several prominent dictionary methods—cBOSS, S-BOSS, WEASEL, TDE, and MrSQM—over 30 resamples of the 106 datasets from the UCR archive for which benchmark results are available for all relevant methods. We rank the accuracy of each method on each dataset, and take the mean rank over all 106 datasets. Lower mean rank corresponds to higher accuracy. Methods for which the pairwise difference in accuracy is not statistically significant, per a Wilcoxon signed-rank test with Holm correction, are connected with a black line (see Demšar 2006; García and Herrera 2008; Benavoli et al. 2016).

Hydra is faster and more accurate than the most accurate existing dictionary methods, TDE and MrSQM, and is competitive in terms of accuracy with several of the most accurate current methods for time series classification: see Sect. 4.1. Hydra can train and test on this subset of 106 datasets in approximately 31 min using a single CPU core, as compared to approximately 1 h 41 min for MrSQM, 4 h for cBOSS, 22 h for TDE, more than 24 h for WEASEL, and more

[1] For a fixed value of $k \times g$, increasing k implies decreasing g, and vice versa.

than 6 days for S-BOSS. Compute times for HYDRA and MrSQM are averages over 30 resamples, run on a cluster using Intel Xeon E5-2680 and Xeon Gold 6150 CPUs, restricted to a single CPU core per dataset per resample. Compute times for cBOSS, TDE, S-BOSS, and WEASEL are taken from Middlehurst et al. (2021a). (These times are only broadly comparable due to hardware and software differences).

HYDRA embodies a connection between conventional dictionary methods and the ROCKET 'family' of methods. At the same time, HYDRA represents a new and highly-effective transform for time series classification in its own right. HYDRA provides an alternative method for performing dictionary-like classification while avoiding the complex apparatus of conventional dictionary methods, and while also being faster and more accurate than existing dictionary methods.

The rest of this paper is structured as follows. In Sect. 2, we present relevant related work. In Sect. 3, we explain HYDRA in detail. In Sect. 4, we present experimental results including results for a number of larger datasets, and a sensitivity analysis for key hyperparameters.

2 Related work

2.1 Dictionary methods

Dictionary (or 'bag of words') methods represent a prominent approach to time series classification. BOSS was identified in Bagnall et al. (2017) (the 'bake off' paper) as one of the three most accurate methods for time series classification on the datasets in the UCR archive (Dau et al. 2019), and was included in the original HIVE-COTE ensemble, then the most accurate method for time series classification on the datasets in the UCR archive (Lines et al. 2018). The most accurate current dictionary methods, TDE (Middlehurst et al. 2021a) and MrSQM (Le Nguyen and Ifrim 2022), are competitive with several of the most accurate current methods for time series classification on the datasets in the UCR archive. TDE is one of the four components of HIVE-COTE 2 (HC2), currently the most accurate method for time series classification on the datasets in the UCR archive (Middlehurst et al. 2021b).

Most dictionary methods work in broadly the same way, i.e., by passing a sliding window over each time series, smoothing or approximating the values in each window, and assigning the resulting values to letters from a symbolic alphabet (Large et al. 2019). The counts of the resulting patterns are used as the basis for classification. BOSS and several other methods use Symbolic Fourier Approximation (SFA) (Schäfer 2015), which involves:

- Passing a sliding window over the input time series;
- Applying the Fourier transform to the values in the window, dropping the high-frequency coefficients (in effect, smoothing the values in the window using a low-pass filter); and
- Assigning the remaining values to one of four letters to form words.

The counts of the resulting words are used to perform classification with a 1-nearest neighbour (1NN) classifier. BOSS is a large ensemble of such classifiers using different hyperparameter configurations.

The resulting feature space is typically very large and very sparse (see Schäfer and Leser 2017; Large et al. 2019; Le Nguyen and Ifrim 2022), and the resulting patterns represent a high degree of approximation, as the input is both smoothed and quantised to a very small set of discrete values. In addition, for methods using SFA or a variation thereof, the patterns are formed over values in the frequency domain, rather than the original input.

Despite the broad similarities between many dictionary methods, different methods can produce very different results due to differences in the way that the input is approximated or quantised (Bagnall et al. 2017; Large et al. 2019). The most prominent dictionary methods are BOSS (Schäfer 2015), cBOSS (Middlehurst et al. 2019), S-BOSS (Large et al. 2019), WEASEL (Schäfer and Leser 2017) and, more recently, TDE (Middlehurst et al. 2021a) and MrSQM (Le Nguyen and Ifrim 2022).

In contrast to BOSS, cBOSS randomly selects hyperparameter combinations for its ensemble, sets an upper limit on ensemble size, and uses a different ensemble weighting. cBOSS is considerably faster than BOSS with approximately the same accuracy on the datasets in the UCR archive.

S-BOSS adds temporal information by recursively dividing the input time series into subseries, and forming dictionaries over the subseries. S-BOSS also uses a different distance measure for the 1NN classifiers. S-BOSS is more accurate than BOSS on the datasets in the UCR archive, but at considerable computational expense.

WEASEL selects Fourier coefficients using a statistical test, performs quantisation based on information gain, and uses a chi-squared test to perform feature selection. WEASEL uses the resulting features to train a logistic regression model. WEASEL is more accurate than BOSS or cBOSS on the datasets in the UCR archive.

TDE incorporates the ensembling method from cBOSS, the temporal information and distance measure from S-BOSS, and the quantisation method from WEASEL. TDE is currently one of the two the most accurate dictionary methods on the datasets in the UCR archive and, as noted above, is one of the four components of HC2.

MrSQM builds on an earlier method, MrSEQL (Le Nguyen et al. 2019). MrSQM uses a combination of random feature selection and feature selection via a chi-squared test, and uses the resulting features to train a logistic regression model. MrSQM is both significantly more accurate, and an order of magnitude faster, than MrSEQL. Along with TDE, MrSQM is one of the two most accurate dictionary methods on the datasets in the UCR archive.

The basic process for extracting discriminative patterns shared by most conventional dictionary methods arguably has some limitations. In particular, transforms such as SFA rely on parameterised smoothing and quantisation operations, representing a large hyperparameter space, and potentially resulting in a very large feature space (even for relatively short patterns and a high degree of discretisation). Different approaches to navigating these issues include covering the hyperparameter space through large ensembles (e.g., BOSS) and/or random hyperparameter selection (e.g., cBOSS and MrSQM), or using statistical approaches to hyperparameter selection (e.g., WEASEL) and/or feature selection (e.g., WEASEL and MrSQM).

However, Hydra shows that there are other, simpler approaches to building dictionaries of discriminative patterns which avoid these peculiarities while producing higher accuracy with lower computational cost.

2.2 Rocket, MiniRocket, and MultiRocket

The Rocket 'family' of methods—namely Rocket and its variants MiniRocket (Dempster et al. 2021) and MultiRocket (Tan et al. 2022)—represent a seemingly quite different approach to time series classification.

Rocket transforms input time series using a large number of random convolutional kernels (by default, 10,000), and uses the transformed features to train a linear classifier. For a given time series, $X = [x_0, x_1, \ldots, x_{n-1}]$, kernel, $W = [w_0, w_1, \ldots, w_{m-1}]$, dilation, d, and bias, b, the convolution operation can be formulated as:

$$X * W - b = \sum_j x_{i+j \cdot d} \cdot w_j - b, \; i \in \{0, \ldots, \hat{n}\}.$$

In this formulation, $\hat{n}$ is the length of the time series adjusted for the length the dilated kernel, i.e., $\hat{n} = n - (m-1) \cdot d - 1$. To this end, it is common practice to append $\lfloor \frac{m}{2} \rfloor \cdot d$ zeros ('padding') to the start and end of each time series, such that the convolution operation begins with the middle element of the kernel centred on the first element of the time series, and ends with the middle element of the kernel centred on the last element of the time series, ensuring that the convolution output has the same length as the input time series.

Rocket uses kernels with lengths selected randomly from $\{7, 9, 11\}$, weights drawn from $\mathcal{N}(0, 1)$, bias values drawn from $\mathcal{U}(-1, 1)$, random dilation (on an exponential scale), and random padding (padding is applied or not with equal probability). Rocket applies both global max pooling, and 'proportion of positive values' (PPV) pooling to the convolution output. For $Z = X * W = [z_0, z_1, \ldots, z_{n-1}]$, and bias, b, PPV is given by $\frac{1}{n} \sum_{i:z_i > b} 1$.

The resulting features are used to train a ridge regression classifier or logistic regression (for larger datasets). The two most important aspects of Rocket in terms of accuracy are the use of dilation and PPV pooling. Rocket achieves state-of-the-art accuracy with a fraction of the computational expense of other methods of comparable accuracy, with the exception of MiniRocket and MultiRocket. Along with TDE, an ensemble of Rocket models, known as Arsenal, is one of the four components of HC2 (Middlehurst et al. 2021b). (We note that the connection between Rocket and dictionary methods demonstrated by Hydra suggests that there is a connection between these two components of HC2).

MiniRocket makes several key changes to Rocket in order to remove randomness and significantly speed up the transform. MiniRocket uses a fixed kernel length of 9, a small, fixed set of 84 kernels, bias values drawn from the convolution output, a fixed set of dilation values (fixed relative to the length of the input time series), and only produces PPV features. (Whereas Rocket uses a single bias value per kernel, MiniRocket uses multiple bias values—producing multiple features—per kernel). MiniRocket uses the

same classifiers as Rocket. MiniRocket is significantly faster than any other method of comparable accuracy, including Rocket, and demonstrates that it is possible to achieve essentially the same accuracy as Rocket using a fixed set of nonrandom kernels.

MultiRocket represents a further extension of Rocket/MiniRocket, adding three additional pooling operations (in addition to PPV), as well as transforming both the original time series and the first-order difference. MultiRocket uses the same kernels and classifiers as MiniRocket. MultiRocket is currently the next-most-accurate method for time series classification after HC2 on the datasets in the UCR archive. MultiRocket achieves broadly similar accuracy to HC2 while being several orders of magnitude faster.

While seemingly very different, we show that there is, in fact, a simple connection between Rocket and dictionary methods, and that this connection provides the basis for performing fast and accurate dictionary-like classification using convolutional kernels.

2.3 Other state of the art

There has been significant progress in terms of accuracy in time series classification since Bagnall et al. (2017). In addition to Rocket and its variants, the most accurate current methods on the datasets in the UCR archive include TDE, MrSQM, DrCIF (Middlehurst et al. 2020, 2021b), InceptionTime (Ismail Fawaz et al. 2020), TS-CHIEF (Shifaz et al. 2020), and HC2 (see Bagnall et al. 2020; Middlehurst et al. 2021b).

DrCIF builds on an earlier method, TSF, to extract multiple features, including 'catch22' features (Lubba et al. 2019), from random intervals. DrCIF takes intervals from the input time series, the first-order difference, and a periodogram. Like TDE and Arsenal, DrCIF is one of the components of HC2.

InceptionTime is an ensemble of deep convolutional neural networks based on the Inception architecture, and is currently the most accurate convolutional neural network model for time series classification on the UCR archive.

TS-CHIEF is an extension of an earlier method, ProximityForest (Lucas et al. 2019), and is an ensemble of decision trees using distance measures, intervals, and spectral splitting criteria.

HC2 supersedes earlier variants of HIVE-COTE (Lines et al. 2018; Bagnall et al. 2020), and is an ensemble comprising TDE, DrCIF, Shapelet Transform, and Arsenal. HC2 is currently the most accurate method for time series classification on the datasets in the UCR archive.

These methods, while highly accurate, are all burdened by high computational complexity. DrCIF takes almost 2 days to train and test on 112 datasets in the UCR archive, TDE more than 3 days, InceptionTime more than 3 days (using GPUs), HC2 approximately 2 weeks, and TS-CHIEF several weeks (Middlehurst et al. 2021b). MrSQM is faster, taking approximately 3 h to train and test on the same datasets using broadly comparable hardware. In comparison, Hydra completes the same task in approximately 41 min.

3 Method

3.1 Overview

Hydra is a dictionary method which uses convolutional kernels, incorporating aspects of both Rocket and conventional dictionary methods.

Hydra involves transforming the input time series using a set of random convolutional kernels, arranged into g groups with k kernels per group, and then at each timepoint counting the kernels representing the closest match with the input time series for each group, the closest match at each timepoint being the kernel with the largest response (i.e., the largest-magnitude dot product with the input at that timepoint): see Fig. 3. The counts are then used to train a linear classifier. (Note that there are g groups *per dilation*).

Like other dictionary methods, Hydra uses patterns that approximate the input, and produces features that represent the counts of those patterns. However, unlike typical dictionary methods, Hydra uses random patterns, represented by random convolutional kernels. Instead of counting exact matches between the input and a set of patterns drawn from the input, Hydra counts the kernel representing the closest match (i.e., having the largest-magnitude dot product) with the input at each timepoint. The level of approximation represented by the closest match is controlled by group size.

More formally, for $\boldsymbol{Z} = X * \boldsymbol{W}$, where X is a time series of length l, and $\boldsymbol{W}$ is a group of k kernels (i.e., such that the i^{th} row of $\boldsymbol{Z}$ corresponds to $X * W_i$), the indices of the kernels representing the closest match with X at each timepoint are given by:

$$U_j = \arg\max_i z_{ij},\ j \in \{0, \ldots, l-1\}.$$

The count of the number of times the ith kernel is the closest match with X is then given by $\sum_{j:u_j=i} 1$.

While the use of random patterns distinguishes Hydra from typical dictionary methods, the difference is not as radical as it might first appear. The patterns extracted by typical dictionary methods represent a considerable degree of approximation of the input time series: see Sect. 2.1. Although they represent a different form of approximation (randomisation, rather than smoothing and quantisation), the patterns identified and counted by Hydra are not necessarily 'more approximate' than the patterns used in typical dictionary methods.

Like Rocket, Hydra transforms the input time series using random convolutional kernels. However, unlike Rocket and its variants: (1) the kernels are organised into groups; and (2) Hydra counts the kernels in each group representing the closest match with the input at each timepoint. In effect, Hydra treats each kernel as a pattern in a dictionary, and treats each group as a dictionary. In a sense, the kernels in each group are forced to 'compete' in order to be counted at each timepoint.

The organisation of the kernels into groups, although seemingly simple, is what allows us to move between models more closely resembling dictionary methods and models more closely resembling Rocket, and has a decisive influence on the accuracy of the method. Further, the move away from producing PPV features represents

a major departure from the ROCKET paradigm. (Recall that for $Z = X * W$, PPV represents the proportion of values above some threshold, b, i.e., $\frac{1}{n}\sum_{i:z_i>b} 1$). PPV pooling is a defining characteristic of the ROCKET 'family' of methods, and one of the most important factors allowing these methods to achieve high accuracy.

The key hyperparameters for HYDRA are:

- *The characteristics of the kernels*—for simplicity, HYDRA largely inherits the characteristics of the kernels from ROCKET and its variants (Sect. 3.2);
- *The way in which the kernels are counted*—by default, at each timepoint HYDRA counts both the kernel with the maximum response (i.e., largest-magnitude positive dot product) and the kernel with the minimum response (i.e., largest-magnitude negative dot product), using both 'hard' and 'soft' counting, as explained below (Sect. 3.3);
- *The number of groups and the number of kernels per group*—by default, HYDRA uses 64 groups ($g = 64$) with 8 kernels per group ($k = 8$), for a total of $k \times g = 512$ kernels per dilation (Sect. 3.4); and
- *Whether not to include the first-order difference*—by default, HYDRA acts on both the original time series as well as the first-order difference (Sect. 3.5).

In setting the values of these hyperparameters, we have restricted ourselves to the same subset of 40 'development' datasets (default training/test splits) from the UCR archive per Dempster et al. (2020, 2021), in order to avoid overfitting the entire archive.

Pseudocode for the HYDRA transform is set out in Appendix D. We implement HYDRA using PyTorch (Paszke et al. 2019). We use the ridge regression classifier from scikit-learn (Pedregosa et al. 2011), and logistic regression implemented using PyTorch. Our code and full results will be made available at https://github.com/angus924/hydra.

3.2 Kernels

HYDRA inherits major kernel characteristics from ROCKET and its variants:

- A kernel length of 9 (per MINIROCKET/MULTIROCKET);
- Weights drawn from $\mathcal{N}(0, 1)$ (per ROCKET); and
- Exponential dilation.

Bias values—which define the threshold above which values in the convolution output are considered 'positive' for the purpose of the PPV features produced by ROCKET and its variants—are not used in HYDRA. As such, we do not have the same flexibility to 'spread' features over different dilations (e.g., by assigning more bias values to smaller or larger dilations), or to increase or decrease the total number of features (e.g., by increasing or decreasing the total number of bias values).

Accordingly, HYDRA uses a simplified set of dilations in the range $\{2^0, 2^1, 2^2, \ldots\}$ (such that the maximum effective length of a kernel including dilation is the length of

the input time series), and does not use a fixed feature count (c.f., approx. 10, 000 for MiniRocket, 20, 000 for Rocket, and approx. 50, 000 for MultiRocket). Additionally, for simplicity, Hydra always uses padding (zeros are appended to the start and end of each time series such that the convolution output is the same length as the input). Using a fixed kernel length, simplified dilation, and consistent padding, greatly simplifies the organisation of kernels into groups. Always padding has the additional advantage of allowing the transform to work by default with variable-length time series.

As noted above, there are g groups *per dilation*. In this sense, dilation acts as a kind of super grouping of kernels. Accordingly, the total number of kernels ($k \times g \times d$, where d is the number of dilations) will grow logarithmically with time series length and, for very long time series, it may be desirable to, e.g., subsample the dilation values, use a smaller number of groups (per dilation), and/or downsample the input. However, we have not used any such restrictions in any of the experiments presented in this paper.

We normalise the kernel weights by subtracting the mean and dividing by the sum of the absolute values (i.e., the L1 norm). This is to ensure that one kernel is not counted over another kernel due to a spurious difference in the mean or magnitude of the kernel weights, i.e., so that the weights for the kernels within each group are in some sense on the same scale. Note that different approaches to normalisation can affect which kernels represent the 'closest match': see Sect. 3.4.2.

It is possible to perform the transform using a smaller set of kernels, e.g., the MiniRocket kernels, and to form the groups by resampling the convolution output, i.e., such that the convolution output for each kernel may appear in more than one group. This could improve speed and scalability, and potentially provide the basis for a deterministic transform. However, initial experimentation suggests that accuracy decreases as the number of unique kernels decreases.

As noted above, MiniRocket and MultiRocket achieve high accuracy without using random kernels. This suggests that it is not necessarily the 'randomness' of the kernels used in Hydra which is important. It is possible that a larger set of MiniRocket-like kernels, or other nonrandom kernels (e.g., kernels generated using a low-discrepancy sequence), would achieve similar accuracy to the random kernels used in Hydra. However, it is not clear whether a fixed grouping or predefined resampling would be effective, or whether the resampling procedure should remain stochastic, even for nonrandom kernels. We leave the exploration of these questions for future work.

3.3 Method of counting

3.3.1 Maximum vs minimum response

By default, at each timepoint Hydra counts both the kernel with the maximum response (output value), i.e., the kernel representing the closest match with the input time series, and the kernel with the minimum response.

Unlike Rocket and its variants, where a kernel is counted in Hydra, the information for the other kernels at that timepoint is discarded. Counting the kernel with the

minimum response, in addition to the kernel with the maximum response, preserves more of the information in the convolution output without requiring additional work in terms of the transform, and improves accuracy: see Sect. 4.3. Note, however, that where $k = 1$, counting both the minimum response and the maximum response is redundant: they are the same.

The kernel with the minimum response does not represent the 'poorest match' to the input time series, but rather an alternative closest match. (The 'poorest match' would be the kernel with the lowest-magnitude response). Counting both the kernel with the maximum response and the kernel with the minimum response is equivalent to looking for the closest match twice: once with the given set of kernels, $\boldsymbol{W}$, and once with the 'inverted' set of kernels, $-\boldsymbol{W}$. (As the kernels are random, the sign of the kernels is arbitrary).

3.3.2 Hard (argmax) vs soft (max) counting

Hydra uses two different forms of counting:

- *'Hard' counting*—incrementing the count of the kernel with the maximum (and/or minimum) response at each timepoint; and
- *'Soft' counting*—accumulating the maximum (and/or minimum) response for the kernel with the max (and/or min) response at each timepoint.

Hard counting involves an argmax (or argmin) operation over the channels in the convolution output for each group (one channel per kernel), returning the index of the kernel with the maximum (or minimum) response at each timepoint. This is then used to increment the counts for the relevant kernels.

Soft counting involves a max (or min) operation over the same channels, returning the maximum (or minimum) response at each timepoint. This is then accumulated for the relevant kernels.

More formally, for $\boldsymbol{Z} = X * \boldsymbol{W}$, where X is a time series of length l, $\boldsymbol{W}$ is a group of k kernels (i.e., such that the ith row of $\boldsymbol{Z}$ corresponds to $X * W_i$), the indices of the kernels representing the closest match with the input time series at each timepoint are given by $U_j = \arg\max_i z_{ij}, j \in \{0, \ldots, l-1\}$, per Sect. 3.1, and the value of the largest response at each timepoint is given by $V_j = \max_i z_{ij}, j \in \{0, \ldots, l-1\}$:

- The 'hard' count for the ith kernel—i.e., the count of the number of times the ith kernel has the largest response—is given by $\sum_{j:u_j=i} 1$; and
- The 'soft' count for the ith kernel—i.e., the sum of the responses for the ith kernel (where it has the largest response)—is given by $\sum_{j:u_j=i} v_j$.

The formulation of V, above, is similar to the 'maxout' function (Goodfellow et al. 2013). Note, however, we do not use V (or U) directly. We do not 'mix' values from different channels. Instead, we pool the values in each channel separately, producing separate features for each kernel.

Although superficially different, both hard and soft counting capture broadly similar information. Soft counts can be expected to be roughly proportional to hard counts, and vice versa. Where a kernel predominates, it will have a higher relative count, and the sum of the response (where that kernel is counted) is also likely to be relatively large.

By default, HYDRA uses both hard counting and soft counting: soft counting for the maximum response, and hard counting for the minimum response. (The allocation of soft counting to the maximum response and hard counting to the minimum response is essentially arbitrary). Using both soft and hard counting is more accurate than using either alone: see Sect. 4.3.2.

3.3.3 Clipping

By default, HYDRA counts the kernel with the maximum response whether or not the maximum response is positive (and, likewise, counts the kernel with the minimum response whether or not the minimum response is negative).

It is possible to 'clip' the values in the convolution output, such that the kernel with the maximum response is only counted when the maximum response is positive (and, likewise, the kernel with the minimum response is only counted when the minimum response is negative). This is essentially equivalent to passing the convolution output through a ReLU function.

Without clipping, the features produced by the most ROCKET-like variant (i.e., $k = 1$) are uninformative:

- Hard counting implies counting every timepoint for every kernel, such that all the features are the same, i.e., a value representing the length of the input time series; and
- Soft counting produces features which are approximately zero, as positive and negative values in the convolution output will 'cancel each other out'.

With clipping, the features produced where $k = 1$ are equivalent to PPV (hard counting), and global average pooling (soft counting): see Sect. 3.4.1. However, clipping is of little practical significance. While it improves accuracy for $k = 1$, it has little or no effect for $k \geq 2$, and has no effect on the optimal values of g and k.

3.4 Number of groups (g) and kernels per group (k)

The number of groups (g) or, equivalently, the number of kernels per group (k), controls the extent to which HYDRA more closely resembles dictionary methods or more closely resembles ROCKET in two senses:

- *The features produced by the transform*—whether the features are more like the features produced by dictionary methods or more like the features produced by ROCKET; and
- *The level of approximation provided by the random kernels*—whether the patterns identified and counted by HYDRA are more, or less, representative of the input time series.

3.4.1 Features

With multiple kernels per group ($k > 1$), HYDRA produces counts like typical dictionary methods. However, as the number of kernels per group decreases, in particular, as the number of kernels per group approaches one ($k \to 1$), the features produced by the transform undergo a qualitative change.

Where $k = 1$, provided that at each timepoint a kernel is only counted if the maximum response is positive (or the minimum response is negative), HYDRA produces PPV and/or global average pooling features:

- Hard counting involves incrementing the count for every kernel wherever the convolution output is positive, i.e., PPV; and
- Soft counting involves accumulating the output values for each kernel wherever the convolution output is positive, i.e., global average pooling.

In other words, where $k = 1$, the features produced by HYDRA no longer represent counts, but instead represent the average response of every kernel across all timepoints in the form of PPV and/or global average pooling. In this sense, $k = 1$ represents the most ROCKET-like variant of HYDRA.

However, as noted above, HYDRA only produces PPV and/or global average pooling features where $k = 1$, and then only where the values in the convolution output are 'clipped'. Additionally, the most ROCKET-like variant of HYDRA is *not* the most accurate variant of HYDRA, whether or not clipping is used. This may be, in part, because the effectiveness of PPV features is closely linked to the use of a large number of bias values and HYDRA, in contrast to ROCKET and its variants, does not use bias values at all.

In this sense, HYDRA can be seen as a kind of generalisation of ROCKET where, instead of applying a threshold to the convolution output for a given kernel, W, in the form of a scalar bias value, b, we instead apply a complex and dynamic threshold in the form of the convolution output for all of the other kernels in each group. That is, for $Z = X * W$, instead of features in the form $\frac{1}{n}\sum_{j:z_j>b} 1$, HYDRA produces features in the form $\sum_{j:z_j>\hat{z}_j} 1$ (hard counting), or $\sum_{j:z_j>\hat{z}_j} z_j$ (soft counting), where $\hat{Z}$ is a threshold defined by the largest value at each timepoint in the convolution output for all other kernels in a group (i.e., not including the given kernel). The convolution output for each kernel, and the features produced by each kernel, are therefore affected by the convolution output for all of the other kernels in each group.

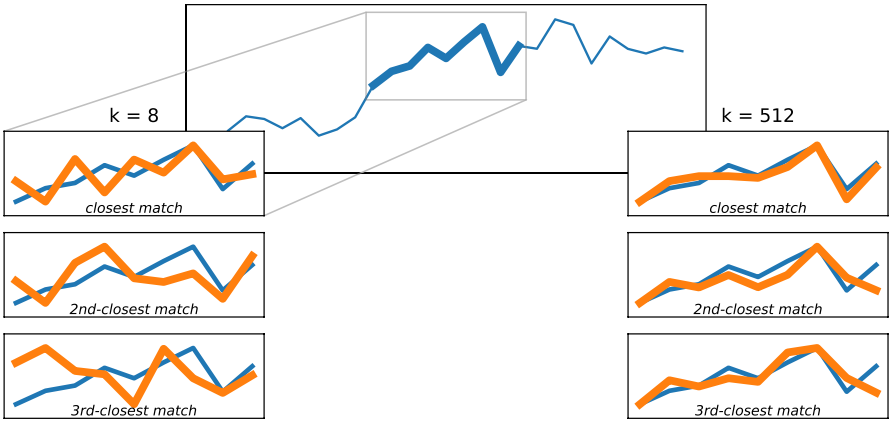

Fig. 4 A time series segment is shown in blue. The kernels representing the three closest matches to that segment (i.e., having the three largest-magnitude dot products with that segment) are shown in orange, for a small ($k = 8$) group, and for a large ($k = 512$) group. A small number of kernels per group (left) implies a high degree of approximation. A large number of kernels per group (right) implies a low degree of approximation

3.4.2 Approximation

A large number of kernels per group (i.e., a small number of groups) implies a low degree of approximation. The larger the number of kernels per group, the more kernels are 'competing' to have the maximum (or minimum) response at each timepoint, and the higher the probability that the kernels identified and counted by HYDRA closely match the input time series: see Fig. 4. (With an infinite number of kernels per group, the kernel with the maximum response at each timepoint would be expected to exactly match the input time series). In this sense, with a larger number of kernels per group, the patterns identified and counted by HYDRA are more like patterns extracted from the input as in typical dictionary methods.

A small number of kernels per group implies a high degree of approximation. The smaller the number of kernels per group, the fewer kernels are 'competing', and the lower the probability that the kernels identified and counted by HYDRA match the input time series. As the number of kernels per group decreases, the patterns identified and counted by HYDRA become increasingly approximate in the sense of becoming increasingly 'random', i.e., unrelated to the input time series.

Further, as the number of kernels per group increases, we observe that feature sparsity also increases, that is, the proportion of features with a value of zero: see Fig. 18, Appendix A. Sparse features correspond to kernels that are never counted at any timepoint. Increasing 'competition' between kernels means that an increasing number of kernels are counted rarely or not at all (i.e., there is at least one other kernel with a larger-magnitude response at every timepoint). In other words, as the number of kernels per group increases, the information carried by the features becomes concentrated in a smaller and smaller number of kernels.

More approximate matches become increasingly redundant (i.e., only kernels which closely match the input at each timepoint will be counted).

As such, for a fixed total number of kernels ($k \times g$), there is a kind of tradeoff between the number of kernels per group and the number of groups. With a larger number of kernels per group (i.e., a smaller number of groups), we are combining information from a smaller number of more precise matches. With a smaller number of kernels per group (i.e., a larger number of groups), we are combining information from a larger number of more approximate matches. It is not clear, a priori, how group size will affect accuracy. Empirically, we find that a large number of small groups produces higher accuracy than a small number of large groups. However, accuracy is broadly similar for groups of size $k \in \{4, 8, 16, 32\}$ for most hyperparameter configurations: see Fig. 12.

Note that the way in which the kernel weights are normalised can affect which kernel represents the 'closest match' at each timepoint (i.e., which kernel has the largest-magnitude dot product with the input at each timepoint). Normalising by the sum of absolute values (i.e., the L1 norm) means that 'winning' kernels tend to have large-magnitude weights matching the smallest and largest values in the input time series in a given window. Normalising by the standard deviation or by the L2 norm means that 'winning' kernels tend to match the overall shape of the input time series in a given window. However, the choice of normalisation appears to have little practical effect on accuracy.

3.5 First-order difference

Like MultiRocket and DrCIF, Hydra operates on both the original input time series and the first-order difference. This is achieved by assigning half of the groups (i.e., by default, 32 of the 64 groups) to operate on the first-order difference. This allows the transform to incorporate the first-order difference without introducing additional features or computation. Adding the first-order difference increases the accuracy of every variant of Hydra: see Sect. 4.3.2. (Note that, as the weights for each kernel sum to zero, by convolving the kernels with the input we are acting implicitly on the first-order difference and, accordingly, by convolving the kernels with the first-order difference, we are acting implicitly on the second-order difference).

3.6 Classifier

Hydra uses the same classifiers as Rocket and its variants, i.e., a ridge regression classifier or logistic regression (for larger datasets, i.e., where the number of training examples is greater than approx. 10,000). In order to limit peak memory usage, by default the transform is performed in batches. As for Rocket and its variants, the Hydra transform can be naturally integrated into the minibatch updates for logistic regression trained using stochastic gradient descent or similar.

3.6.1 Feature normalisation

As noted above, as the number of kernels per group increases, we observe increasing feature sparsity. This makes it increasingly difficult to estimate quantities such as the per-feature standard deviation typically used to perform feature normalisation prior to classifier training. Accordingly, we modify the standard approach of subtracting the per-feature mean and dividing by the per-feature standard deviation to take into account feature sparsity, as detailed in Appendix B.

This modified approach to feature normalisation significantly improves accuracy for larger group sizes: compare Figs. 12 and 21. It also modestly improves accuracy for smaller group sizes: see Sect. 4.3.4.

3.7 Limitations

While Hydra has some advantages over conventional dictionary methods and other methods for time series classification, it also has certain limitations.

The large number of groups, and the large number of features produced by the transform, both hinder interpretability. The decision boundary learned by the classifier is a function of a large number of features representing complex interactions between a large number of competing patterns, in multiple groups, across multiple dilations, for both the input time series and the first-order difference. One possible approach to interpretability would be to consider the way in which different groups of kernels segment different time series. We can identify each value in each time series by the index of the 'winning' kernel at that timepoint, and then consider the resulting segments of each timeseries (i.e., where each segment consists of a contiguous set of values with the same 'winning' kernel). However, we leave this for future work.

Additionally, the use of a fixed set of patterns, and the use of a linear classifier, must limit overall model capacity compared to learned features and nonlinear classifiers. In the context of larger datasets, we would expect Hydra to asymptote at a lower accuracy than a model with learned features such as a conventional convolutional neural network. Nevertheless, the accuracy of Hydra is still improving even after approximately 125,000 training examples on the *MosquitoSound* dataset: see Fig. 10.

3.8 Complexity

3.8.1 Computational complexity

Hydra involves convolving the input time series with a set of kernels, performing max/argmax and min/argmin operations on the resulting convolution output, and counting the 'winning' kernels. The computational cost of each of these operations is proportional to the total number of kernels.

As set out in Sect. 3.2, above, Hydra uses $k \times g$ kernels per dilation. The number of dilations grows logarithmically with time series length. Accordingly, the computational cost of the transform is $O(n \cdot d \cdot k \cdot g \cdot l)$, where n is the number of input time series, d is the number of dilations (proportional to $\log_2 l$), k is the number of kernels per group, g is the number of groups, and l is the length of the input time series (assuming that all time series are the same length).

3.8.2 Memory complexity

Hydra temporarily stores the convolution output in order to perform the max/argmax, min/argmin, and counting operations. However, the transform is performed separately for each dilation (the convolution output for a given dilation is 'abandoned' after the counting operation is completed) and both training and test examples can be transformed in batches of any size (from one to the size of the training or test set). The minimum memory requirement for the transform is therefore proportional to the size of the convolution output for a single time series for a single dilation, i.e., $O(k \times g \times l)$, where k is the number of kernels per group, g is the number of groups, and l is the length of the input time series. By default, the transform is performed in batches of 256 examples in order to maintain a constant memory overhead, i.e., the memory requirement is fixed, and is not proportional to the number of examples.

4 Experiments

We evaluate Hydra on the datasets in the UCR archive (Sect. 4.1), demonstrating that Hydra is more accurate than the most accurate existing dictionary methods, TDE and MrSQM, and achieves similar accuracy to several of the most accurate current methods for time series classification. We further demonstrate the accuracy of Hydra on a number of larger datasets (Sect. 4.2). We also explore the effect of key hyperparameters in terms of the number of groups (g) and kernels per group (k), the way in which the kernels are counted, whether or not to include the first-order difference, whether or not to 'clip' the convolution output, and whether or not to apply robust feature normalisation (Sect. 4.3).

4.1 UCR archive

We evaluate Hydra on the datasets in the UCR archive (Dau et al. 2019). We compare Hydra against several prominent dictionary methods, and against the most accurate current methods for time series classification on the datasets in the UCR archive, namely, TDE, MrSQM, DrCIF, Rocket and its variants, InceptionTime, TS-CHIEF, and HC2. For direct comparability with other published results, we evaluate Hydra on the same 30 resamples of 112 datasets from the UCR archive per Middlehurst et al. (2020, 2021a, b). The total compute time for Hydra on all 112 datasets, averaged over 30 resamples, is approximately 41 min using a single CPU core.

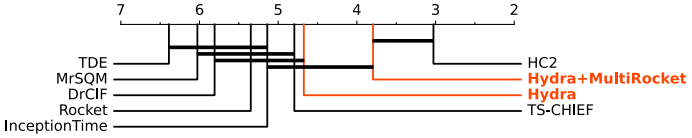

Fig. 5 Mean rank of HYDRA and HYDRA+MULTIROCKET in terms of accuracy versus other SOTA methods over 30 resamples of 112 datasets from the UCR archive

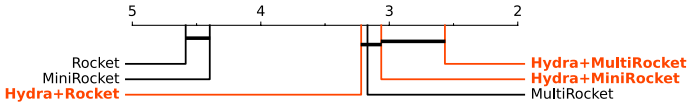

Fig. 6 Mean rank of ROCKET and its variants, with and without HYDRA, over the same 30 resamples of 112 datasets

Figure 1 (page 1) shows the mean rank of HYDRA versus several prominent dictionary methods, namely, cBOSS, S-BOSS, WEASEL, TDE, and MrSQM. HYDRA is faster and, on average, more accurate than all other dictionary methods, including TDE and MrSQM. This ranking is performed on only 106 of the 112 datasets, as results for S-BOSS and WEASEL are unavailable for the six largest datasets due to computational constraints (see Middlehurst et al. 2021a). We have obtained the results for MrSQM using the implementation available at https://github.com/mlgig/mrsqm, with the hyperparameter configuration as set out in Le Nguyen and Ifrim (2022).

Figure 5 shows the mean rank of HYDRA and HYDRA+MULTIROCKET versus the most accurate current methods for time series classification over 30 resamples of 112 datasets from the UCR archive. (The p values corresponding to the pairwise Wilcoxon signed-rank tests between HYDRA, HYDRA+MULTIROCKET, and each of the other classifiers, are provided in Fig. 22, Appendix A). HYDRA is combined with MULTIROCKET by simply concatenating the features produced by each transform. That is, we transform the input time series using HYDRA and, separately, using MULTIROCKET, and then concatenate the two sets of features (and the concatenated features are then used to train the classifier). On average, HYDRA is more accurate (i.e., is more accurate on more than half the datasets) than TDE, MrSQM, and DrCIF, and achieves broadly similar accuracy to ROCKET, InceptionTime, and TS-CHIEF, but is less accurate than HC2. On average, HYDRA+MULTIROCKET is not significantly less accurate than HC2 (with a p value of $0.1068 > 0.05$ for the Wilcoxon signed-rank test indicating no statistically-significant difference, whether or not the Holm correction is applied).

Figure 6 shows the mean rank of ROCKET and its variants, with and without HYDRA. (The pairwise differences between MULTIROCKET, HYDRA+ROCKET, and HYDRA+MINIROCKET are not statistically significant. Similarly, the pairwise difference between HYDRA+MINIROCKET and HYDRA+MULTIROCKET is not statistically significant). The addition of HYDRA significantly improves the accuracy of ROCKET, MINIROCKET, and MULTIROCKET. This is consistent with results for a number of larger

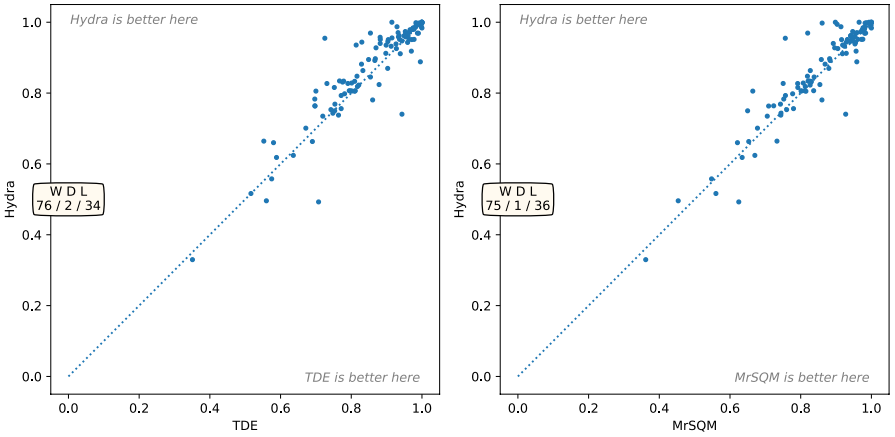

Fig. 7 Pairwise accuracy of Hydra vs TDE (left) and MrSQM (right)

datasets: see Sect. 4.2. Hydra+Rocket and Hydra+MiniRocket both achieve similar accuracy to MultiRocket. While Hydra+MultiRocket 'wins' on almost three times as many datasets as MultiRocket (with a win/draw/loss of 76/9/27), the magnitude of the differences in accuracy are mostly very small. This is not necessarily surprising, as MultiRocket is already one of the two most accurate methods for time series classification on the datasets in UCR archive, and already produces a large number of features.

Figure 7 shows the pairwise accuracy of Hydra versus the two most accurate existing dictionary methods for time series classification: TDE (left) and MrSQM (right). Hydra is more accurate than TDE on 76 datasets, and less accurate on 34. Similarly, Hydra is more accurate than MrSQM on 75 datasets, and less accurate on 36.

Figure 8 shows the pairwise accuracy of Hydra versus Rocket (left), and an alternative configuration of Hydra using a single group (right). (This is the most dictionary-like variant of Hydra, using a single group and hard counting). Hydra is more accurate than Rocket on 64 datasets, and less accurate on 45. The default variant of Hydra ($k = 8/g = 64$) is more accurate than the most dictionary-like variant of Hydra ($k = 512/g = 1$) on 91 datasets, and less accurate on 18.

Figure 9 shows the pairwise accuracy of Hydra (left) and Hydra+MultiRocket (right) versus HC2. Hydra is more accurate than HC2 on 35 datasets and less accurate on 72. Hydra+MultiRocket is more accurate than HC2 on 47 datasets and less accurate on 59 (compared to 42/66 for Hydra+MiniRocket and 42/66 for Hydra+Rocket).

While the pairwise comparison between Hydra and HC2 is heavily in favour of HC2, the actual differences in accuracy are mostly relatively small. In contrast, the difference in computational cost is enormous. The magnitude of the pairwise difference between Hydra and HC2 is: less than 5% for 92% of datasets (93% for Hydra+MultiRocket); less than 2.5% for 78% of datasets (84% for Hydra+MultiRocket); and less than 1% for 55% of datasets (55% for

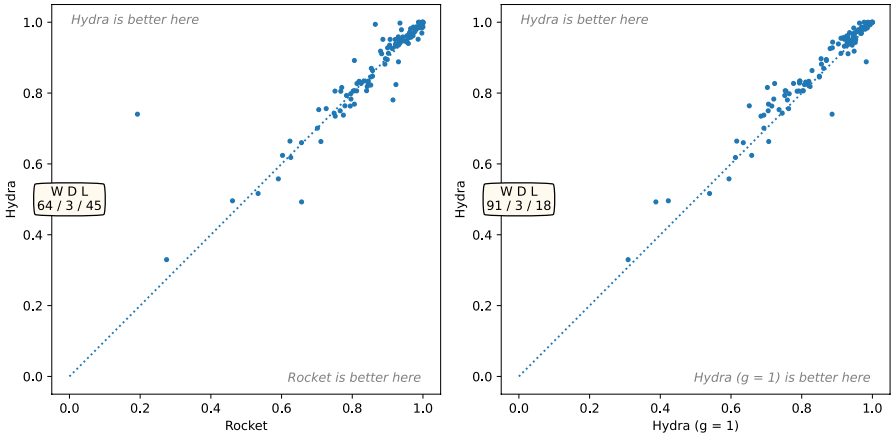

Fig. 8 Pairwise accuracy of HYDRA vs ROCKET (left) and HYDRA ($g = 1$) (right)

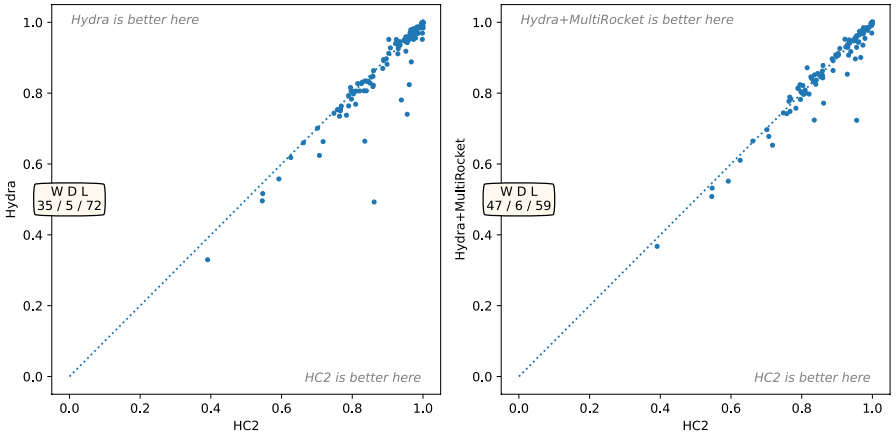

Fig. 9 Pairwise accuracy of HYDRA (left) and HYDRA+MULTIROCKET (right) vs HC2

HYDRA+MULTIROCKET). However HYDRA, like ROCKET and its variants, requires only a small fraction of the compute time of other methods of comparable accuracy: less than 1% of the compute time of TDE, and approximately 0.2% of the compute time of HC2. In other words, the advantages of HC2 over HYDRA in terms of accuracy come at a 500× computational cost.

4.2 Larger datasets

We demonstrate the accuracy and scalability of HYDRA on the three largest datasets in the UCR archive, namely *MosquitoSound* (139,780 training examples, each of length 3750), *InsectSound* (25,000 training examples, each of length 600), and *FruitFlies* (17,259 training examples, each of length 5000). These newer additions

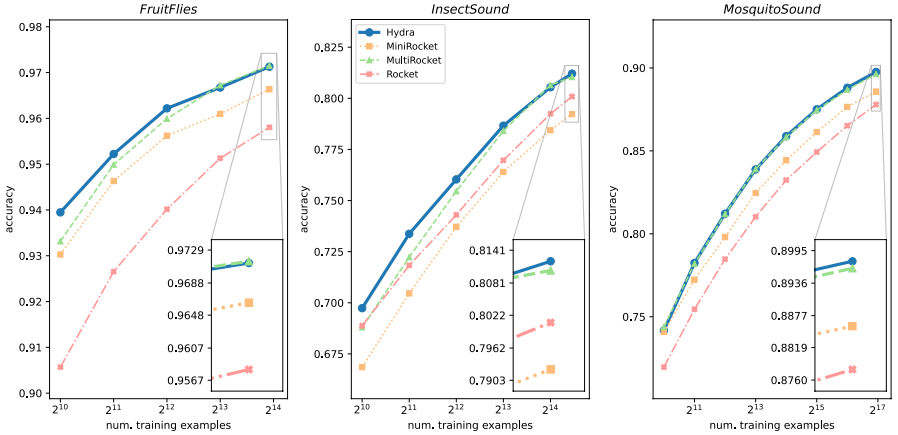

Fig. 10 Learning curves for HYDRA versus MINI/MULTI/ROCKET

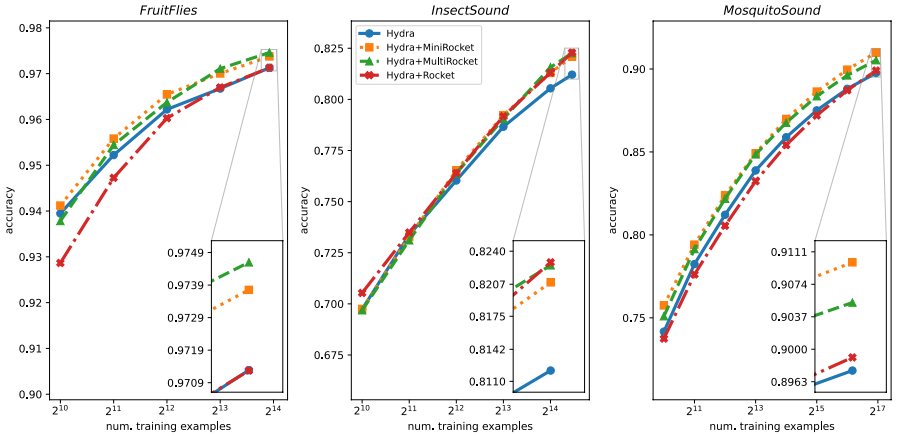

Fig. 11 Learning curves for HYDRA versus HYDRA+MINI/MULTI/ROCKET

to the archive are separate from the 112 datasets referred to in earlier experiments, and are significantly larger than the other datasets in the archive.

For this purpose, we integrate HYDRA and the ROCKET 'family' of methods with logistic regression trained using stochastic gradient descent (Smith, 2017). Full training details are provided in Appendix C.

Learning curves for HYDRA as well as ROCKET, MINIROCKET, and MULTIROCKET are shown in Fig. 10. Learning curves for the combination of HYDRA with each of ROCKET, MINIROCKET, and MULTIROCKET are shown in Fig. 11. As noted above, HYDRA is combined with the other methods by simply concatenating the features generated by each transform.

HYDRA is more accurate than either ROCKET or MINIROCKET on all three datasets, demonstrating similar accuracy to MULTIROCKET in each case. (We note that

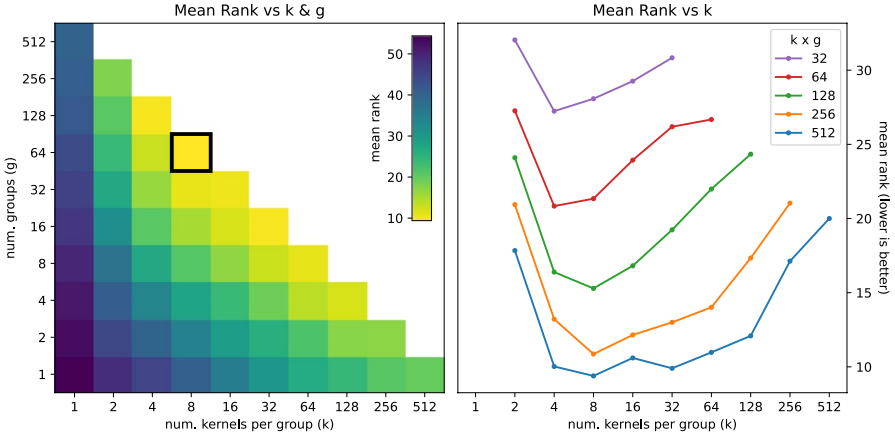

Fig. 12 Mean rank (accuracy) versus the number of kernels per group (k), and the number of groups (g)

Hydra achieves this accuracy with approximately one fifth of the features produced by MultiRocket). Combining Hydra with Rocket, MiniRocket, or MultiRocket meaningfully improves the accuracies of each of these methods.

Interestingly, the accuracy of all methods is continuing to improve at the largest training set sizes. Accuracy is still improving after approximately 125, 000 training examples in the case of the *MosquitoSound* dataset.

Training times are shown in Fig. 23, in Appendix C. As expected, MiniRocket is the fastest method in all cases, being more than 5× faster than any other method on the largest dataset. Hydra is marginally slower than MultiRocket on the largest dataset. Note that the absolute timings are somewhat arbitrary, as all methods can be run using more or fewer cores, and all methods can also be adapted to run on GPU.

4.3 Sensitivity analysis

We explore the effect of key hyperparameters in terms of:

- The number of groups (g), and the number of kernels per group (k);
- The way in which the kernels are counted;
- Whether or not to include the first-order difference;
- Whether or not to 'clip' the convolution output; and
- Whether or not to apply robust feature normalisation.

We conduct the sensitivity analysis using the subset of 40 'development' datasets (default training/test splits) from the UCR archive per Dempster et al. (2020, 2021): see Sect. 3. Results are mean results over 10 runs.

4.3.1 Number of groups (g) and kernels per group (k)

Figure 12 shows the relationship between accuracy and the number of groups (g), and kernels per group (k), for the most accurate variant of Hydra (counting max and min response, using soft and hard counting, including first-order difference, no clipping, and robust normalisation: see Sects. 4.3.2, 4.3.3, and 4.3.4). Accuracy is represented by mean rank: lower mean rank corresponds to higher accuracy. For each dataset, we rank the accuracy of each combination of k and g, then take the mean rank over all datasets.

Figure 12 (left) shows that 64 groups of 8 kernels ($k = 8/g = 64$) produces the highest accuracy. Interestingly, this result is consistent across most (although not all) hyperparameter configurations: counting the maximum response or both the maximum response and minimum response, soft, hard, or both soft and hard counting, with or without clipping, and with or without robust feature normalisation. Additional results for the same model without the first-order difference, with clipping, and without robust feature normalisation, are shown in Appendix A.

Figure 12 (right) shows the same mean ranks versus k. Each line represents a different total number of kernels. This shows that, except for a small total number of kernels, the optimal value of k is approximately 8, suggesting that for any given 'budget' ($k \times g$), it is more effective to increase the number of groups, keeping the number of kernels per group at approximately 8.

This also shows that, in general, adding more kernels increases accuracy. For any number of kernels per group (k), a larger number of groups (g)—in other words, a larger total number of kernels ($k \times g$)—is more accurate.

The number of kernels represents a tradeoff between accuracy and computational efficiency. While increasing the total number of kernels tends to increase accuracy, the magnitude of the improvement decreases as the number of kernels increases. Although there is a clear difference in rank between $k \times g = 256$ and $k \times g = 512$, the actual differences in accuracy are small: see Fig. 16, Appendix A. Increasing the total number of kernels in order to meaningfully increase accuracy beyond $k \times g = 512$ (e.g., $k \times g = 1024$) would involve considerable additional computational burden for little practical gain.

Additionally, limiting the total number of kernels per dilation ($k \times g$) to 512 means that the total number of features per dataset is less than 10, 000 (at least for the 40 'development' datasets) or, in other words, roughly commensurate with MiniRocket. For these reasons, and given that Hydra is already approximately as accurate as Rocket, we set $k \times g = 512$ as the maximum number of kernels per dilation.

Different values of k represent different levels of approximation in the kernels: see Sect. 3.4.2. Figure 12 shows that a relatively high degree of approximation is desirable, broadly consistent with the high level of approximation typical of dictionary methods. The most accurate variants of Hydra are high-g/low-k models, i.e., using a large number of small groups rather than a small number of large groups. This implies that it is preferable to combine the information from many approximate matches. A large number of small groups also has the benefit of low variability: see Fig. 17, Appendix A.

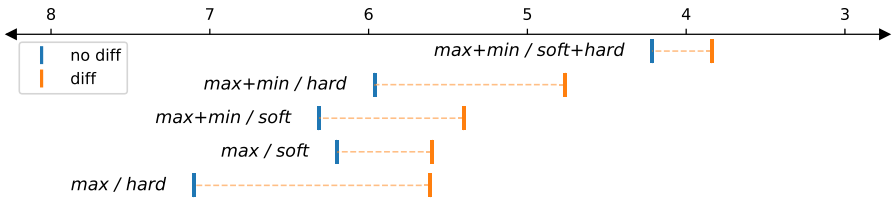

Fig. 13 Mean rank of different variants of Hydra, with and without first-order difference

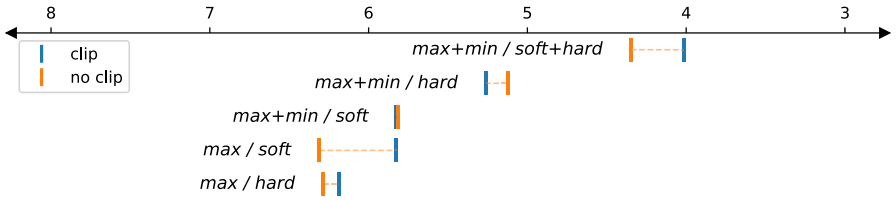

Fig. 14 Mean rank of different variants of Hydra, with and without clipping

We also observe that, as k increases, there is increasing sparsity, i.e., an increasing proportion of kernels which are never counted: see Fig. 18, Appendix A. For the optimal combination of k and g (i.e., $k = 8/g = 64$), Hydra produces dense features.

4.3.2 Method of counting and first-order difference

Figure 13 shows the mean rank of different variants of Hydra, with and without the first-order difference (for the most accurate combination of k/g in each case, and in all cases not using clipping). Figure 13 shows that counting both the maximum and minimum responses is more accurate than only counting the maximum response (at least when including the first-order difference), that using both soft and hard counting is more accurate than using either soft or hard counting alone, and that including the first-order difference always improves accuracy.

4.3.3 Clipping

Figure 14 shows the mean rank of different variants of Hydra, with and without clipping (for the most accurate combination of k/g, being $k = 8/g = 64$ in all cases, and in all cases including the first-order difference). Figure 14 shows that clipping has a small and inconsistent effect on accuracy for optimal values of k/g. In fact, clipping has essentially no effect for $k \geq 2$. The differences in rank shown in Fig. 14 are an artefact of extremely small differences in accuracy (the mean difference in accuracy between clipping and not clipping for the most accurate model is less than 0.0002).

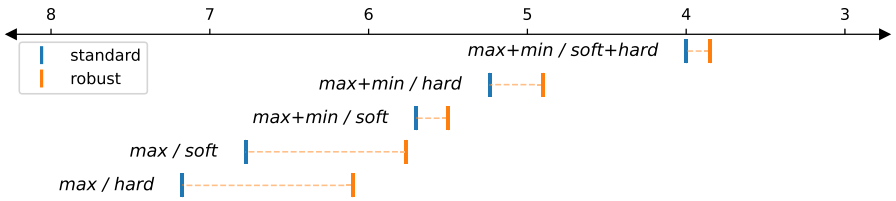

Fig. 15 Mean rank for variants of Hydra with and without robust feature normalisation

4.3.4 Feature normalisation

Figure 15 shows the mean rank of different variants of Hydra, with and without robust feature normalisation (for the most accurate combination of k/g in each case, and in each case including the first-order difference). Figure 15 shows that robust feature normalisation provides a small but consistent benefit in terms of accuracy, even for smaller group sizes.

5 Conclusion

We demonstrate a simple connection between dictionary methods for time series classification, which extract and count symbolic patterns in time series, and methods based on transforming the input time series using convolutional kernels, namely Rocket. This provides the basis for fast and accurate dictionary-like classification using convolutional kernels. We present Hydra, a simple, fast, and highly-accurate dictionary method for time series classification, incorporating aspects of both Rocket and conventional dictionary methods. Hydra demonstrates the advantages of performing dictionary-like classification with convolutional kernels and random patterns. Hydra is faster and more accurate than the most accurate existing dictionary methods, and significantly improves the accuracy of Rocket and its variants when used in combination with these methods. The differences in accuracy between Hydra+MultiRocket and HC2 on the datasets in the UCR archive are not statistically significant, despite Hydra+MultiRocket requiring less than 0.5% of the compute time of HC2. Hydra scales linearly with the number of training examples, and is as accurate or more accurate than MultiRocket on the three largest datasets in the UCR archive. In future work we plan to explore deterministic variants of Hydra, more sophisticated approaches to combining Hydra with other methods, and the application of Hydra to multivariate time series.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s10618-023-00939-3.

Acknowledgements This material is based on work supported by an Australian Government Research Training Program Scholarship and the Australian Research Council under award DP210100072. 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 Ismail Fawaz et al. (2019).

Funding Open Access funding enabled and organized by CAUL and its Member Institutions.

References

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

Bagnall A, Flynn M, Large J et al (2020) On the usage and performance of the hierarchical vote collective of transformation-based ensembles version 1.0 (HIVE-COTE v1.0). In: Lemaire V, Malinowski S, Bagnall A et al (eds) Advanced analytics and learning on temporal data. Springer, Cham, pp 3–18

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

Dau HA, Bagnall A, Kamgar K et al (2019) The UCR time series archive. IEEE CAA J Autom Sin 6(6):1293–1305

Dempster A, Petitjean F, Webb GI (2020) Rocket: exceptionally fast and accurate time series classification using random convolutional kernels. Data Min Knowl Disc 34(5):1454–1495

Dempster A, Schmidt DF, Webb GI (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. ACM, New York, pp 248–257

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

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

Goodfellow I, Warde-Farley D, Mirza M et al (2013) Maxout networks. In: Dasgupta S, McAllester D (eds) Proceedings of the 30th international conference on machine learning, pp 1319–1327

Ismail Fawaz H, Forestier G, Weber J et al (2019) Deep learning for time series classification: a review. Data Min Knowl Discov 33(4):917–963

Ismail Fawaz H, Lucas B, Forestier G et al (2020) InceptionTime: finding AlexNet for time series classification. Data Min Knowl Discov 34(6):1936–1962

Large J, Bagnall A, Malinowski S et al (2019) On time series classification with dictionary-based classifiers. Intell Data Anal 23(5):1073–1089

Le Nguyen T, Ifrim G (2022) Fast time series classification with random symbolic subsequences. In: 7th workshop on advanced analytics and learning on temporal data

Le Nguyen T, Gsponer S, Ilie I et al (2019) Interpretable time series classification using linear models and multi-resolution multi-domain symbolic representations. Data Min Knowl Discov 33(4):1183–1222

Lines J, Taylor S, Bagnall A (2018) Time series classification with HIVE-COTE: the hierarchical vote collective of transformation-based ensembles. ACM Trans Knowl Discov Data. https://doi.org/10.1145/3182382

Lubba CH, Sethi SS, Knaute P et al (2019) catch22: CAnonical Time-series CHaracteristics. Data Min Knowl Discov 33(6):1821–1852

Lucas B, Shifaz A, Pelletier C et al (2019) Proximity Forest: an effective and scalable distance-based classifier for time series. Data Min Knowl Disc 33(3):607–635

Middlehurst M, Vickers W, Bagnall A (2019) Scalable dictionary classifiers for time series classification. In: Yin H, Camacho D, Tino P et al (eds) Intelligent data engineering and automated learning. Springer, Cham, pp 11–19

Middlehurst M, Large J, Bagnall A (2020) The Canonical Interval Forest (CIF) classifier for time series classification. In: IEEE international conference on big data, pp 188–195

Middlehurst M, Large J, Cawley G et al (2021a) The temporal dictionary ensemble (TDE) classifier for time series classification. In: Hutter F, Kersting K, Lijffijt J et al (eds) Machine learning and knowledge discovery in databases. Springer, Cham, pp 660–676

Middlehurst M, Large J, Flynn M et al (2021b) HIVE-COTE 2.0: a new meta ensemble for time series classification. Mach Learn 110:3211–3243

Paszke A, Gross S, Massa F et al (2019) PyTorch: an imperative style, high-performance deep learning library. In: Wallach H, Larochelle H, Beygelzimer A et al (eds) Advances in neural information processing systems 32, pp 8024–8035

Pedregosa F, Varoquaux G, Gramfort A et al (2011) Scikit-learn: machine learning in Python. J Mach Learn Res 12:2825–2830

Schäfer P (2015) The BOSS is concerned with time series classification in the presence of noise. Data Min Knowl Discov 29(6):1505–1530

Schäfer P, Leser U (2017) Fast and accurate time series classification with WEASEL. In: Proceedings of the 2017 ACM on conference on information and knowledge management. ACM, New York, pp 637–646

Shifaz A, Pelletier C, Petitjean F et al (2020) TS-CHIEF: a scalable and accurate forest algorithm for time series classification. Data Min Knowl Discov 34(3):742–775

Smith LN (2017) Cyclical learning rates for training neural networks. In: IEEE winter conference on applications of computer vision, pp 464–472

Tan CW, Dempster A, Bergmeir C et al (2022) MultiRocket: multiple pooling operators and transformations for fast and effective time series classification. Data Min Knowl Discov 36(5):1623–1646