A Supplementary Figures

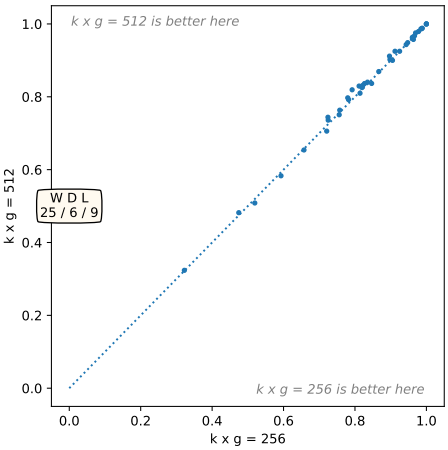

Fig. 16 Pairwise accuracy of $k \times g = 512$ vs $k \times g = 256$.

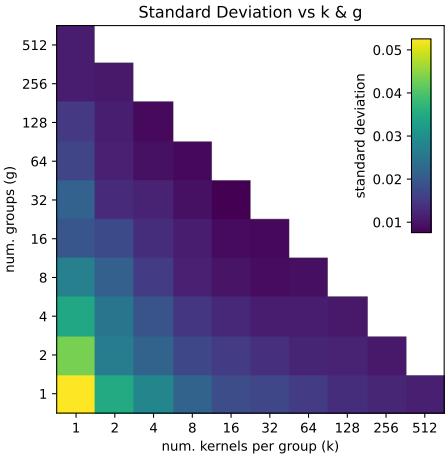

Fig. 17 Standard deviation (accuracy) vs k & g.

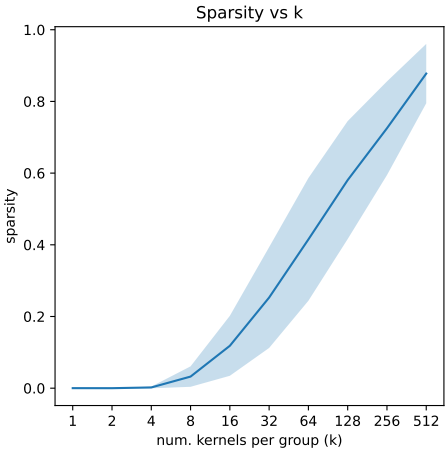

Fig. 18 Sparsity vs k. Sparsity represents the proportion of all features, for all examples in the training set, which have a value of zero (i.e., corresponding to kernels which have not been counted at any timepoint).

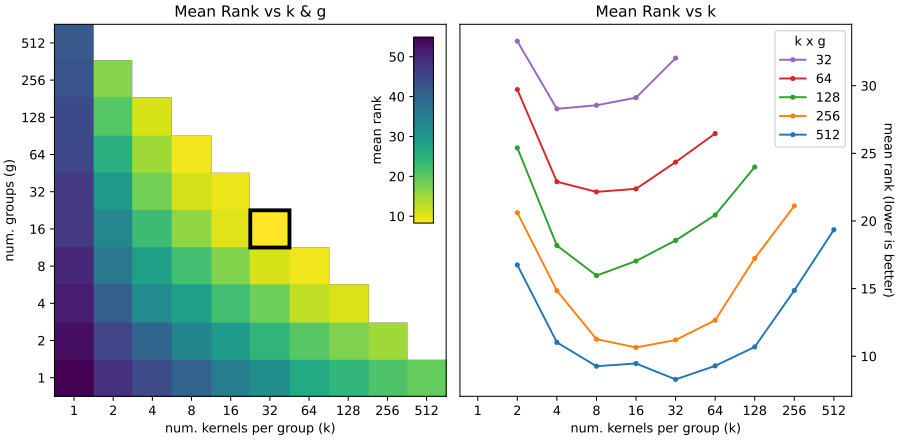

Fig. 19 Mean rank vs k & g (*max+min/soft+hard/no diff/no clip*).

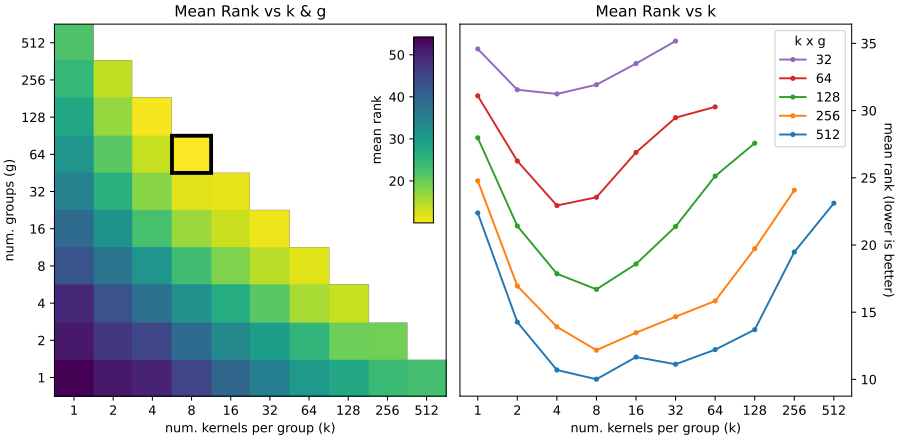

Fig. 20 Mean rank vs k & g (*max+min/soft+hard/diff/clip*).

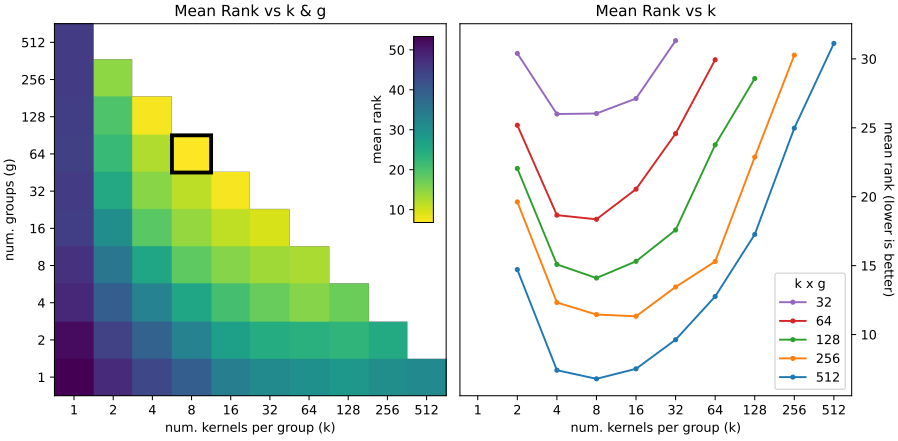

Fig. 21 Mean rank vs k & g (*max+min/soft+hard/diff/no clip/without robust feature normalisation*).

	HC2	Hydra+Multi	Hydra	TS-CHIEF	ITime	Rocket	DrCIF	MrSQM	TDE
Hydra+Multi	0.1068	—	**0.0006**	**0.0026**	**0.0060**	**0.0000**	**0.0000**	**0.0000**	**0.0000**
Hydra	**0.0000**	**0.0006**	—	0.9977	0.7983	0.1059	**0.0030**	**0.0009**	**0.0001**

Fig. 22 p values for the Wilcoxon signed-rank test for each pair of classifiers, corresponding to the results shown in Figure 5. p values less than 0.05 are shown in bold, p values which are considered statistically significant after applying the Holm correction are underlined.

B Feature Normalisation

As the number of kernels per group increases, we observe increasing feature sparsity. With a high level of sparsity, all or almost all of the values for a particular feature might be zero. The sample size (i.e., the number of nonzero values for a given feature) may be too small to meaningfully estimate the standard deviation or any other quantity we might use to rescale the features. A particular problem can arise where we *underestimate* a quantity such as the standard deviation, which can lead to considerable inflation of feature values.

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. In particular, we (a) mask sparse values (these remain zero), (b) square-root transform the features, (c) subtract the per-feature mean, and (d) divide by the per-feature standard deviation plus a quantity, ϵ, which is proportional to per-feature sparsity:

$$\hat{\boldsymbol{Z}} = \frac{\sqrt{\boldsymbol{Z}} - \mu_{\sqrt{\boldsymbol{Z}}}}{\sigma_{\sqrt{\boldsymbol{Z}}} + \epsilon}.$$

Specifically, $\epsilon = \alpha^4$, where α represents per-feature sparsity, i.e., the proportion of values which are zero for each feature. Accordingly, ϵ is in the range $[0, 1]$, and the exponent means that ϵ remains small for small values of sparsity, increasing rapidly as sparsity increases. (Note that the actual value of the exponent is not critically important, and values between 2 and 16 appear to produce broadly similar results.)

C Larger Datasets

C.1 Training Times

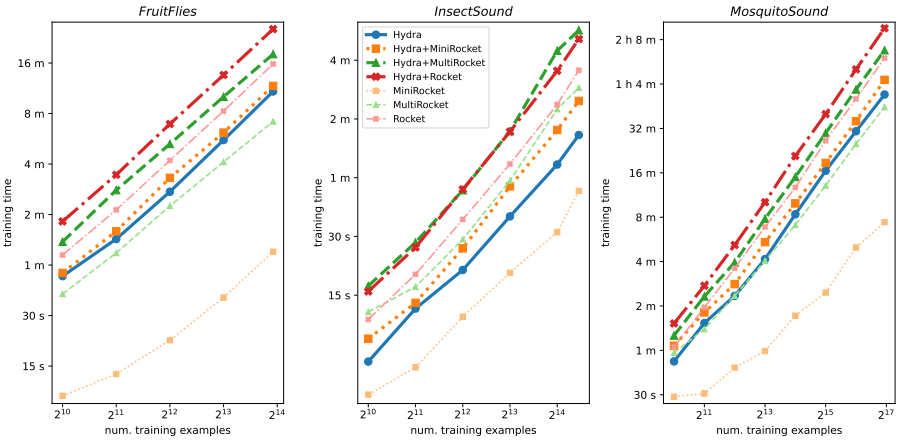

Fig. 23 Training times (not including the transform time for the validation set) for the *FruitFlies*, *InsectSound*, and *MosquitoSound* datasets.

C.2 Training Details

We train a logistic regression model using stochastic gradient descent, with a momentum value of 0.9, and a minibatch size of 256. We use a validation set of 10% of the training examples. We integrate a version of the 'learning rate range test' or 'learning rate finder' into the start of training (Smith, 2017). We increase the learning rate (from an initial value of 10^{-6}) by 10% on each update until validation loss diverges. We then 'rewind' training to the point of minimum validation loss and continuing training. We reduce the learning rate by a factor of 10 if validation loss does not improve after 10 epochs, and stop training if validation loss does not improve after 20 epochs.

For MINIROCKET and MULTIROCKET, we fit the bias values using the first 4,096 (shuffled) training examples. For all methods, we perform the transform once and cache the transformed features in order to avoid repeating the transform unnecessarily. We run the experiments on the same cluster referred to in Section 1, performing five runs per datasets per method (results are mean results over the five runs), using eight cores per dataset per run.

D Pseudocode

Function fit($\boldsymbol{X}$, g, k)

input : $\boldsymbol{X}$: n time series of length l
g : num. groups
k : num. kernels per group

output: $\boldsymbol{W}$: kernels
m : num. dilations

$m \leftarrow \lceil \log_2(l-1)/8 \rceil$ // num. dilations

$h \leftarrow \lfloor g/\min(g,2) \rfloor$ // half num. groups (if $g > 1$)

// $k \times g$ kernels of length 9 per dilation ($h \times k$ for X; $h \times k$ for diff(X), if $g > 1$)

$\boldsymbol{W} \leftarrow \text{sample}(\mathcal{N}(0,1), [m, \min(g,2), h \times k, 9])$

// for each kernel, subtract mean and divide by sum of absolute values

rescale($\boldsymbol{W}$)

return $\boldsymbol{W}$, m

Function transform($\boldsymbol{X}$, $\boldsymbol{W}$, m, g, k)

input : $\boldsymbol{X}$: n time series of length l
$\boldsymbol{W}$: kernels
m : num. dilations
g : num. groups
k : num. kernels per group

output: $\boldsymbol{F}$: features

$h \leftarrow \lfloor g/\min(g,2) \rfloor$ // half num. groups (if $g > 1$)

for $X \in \boldsymbol{X}$ **do** // for each timeseries

  if $g > 1$ **then** $\Omega \leftarrow \{X, \text{diff}(X)\}$ **else** $\Omega \leftarrow \{X\}$ // time series, 1st diff (if $g > 1$)

  for $p \in \{0, \ldots, |\Omega| - 1\}$ **do** // for each of X, diff(X) (if $g > 1$)

    for $d \in \{2^0, \ldots, 2^{m-1}\}$ **do** // for each dilation

      for $q \in \{0, \ldots, h-1\}$ **do** // for each group

        $\boldsymbol{Z} \leftarrow \Omega_p * \boldsymbol{W}^{(d,p,q)}$ // convolve w/ corresponding kernels

        $\check{U}, \check{V}, \hat{U} \leftarrow \mathbf{0}_l, \mathbf{0}_l, \mathbf{0}_l$ // initialise argmax, max, argmin

        for $j \in \{0, \ldots, l-1\}$ **do** // for each timepoint

          $\check{U}_j \leftarrow \arg\max_i z_{ij}$ // index of kernel with max value

          $\check{V}_j \leftarrow \max_i z_{ij}$ // max value

          $\hat{U}_j \leftarrow \arg\min_i z_{ij}$ // index of kernel with min value

        $\check{C}, \hat{C} \leftarrow \mathbf{0}_k, \mathbf{0}_k$ // initialise counts

        for $i \in \{0, \ldots, k-1\}$ **do** // for each kernel

          $\check{C}_i \leftarrow \sum_{j:\check{u}_j = i} \check{v}_j$ // soft count (max)

          $\hat{C}_i \leftarrow \sum_{j:\hat{u}_j = i} 1$ // hard count (min)

        append($\boldsymbol{F}$, $\{\check{C}, \hat{C}\}$) // append counts to features

return $\boldsymbol{F}$