Highly Scalable Time Series Classification for Very Large Datasets

Angus Dempster, Chang Wei Tan, Lynn Miller,
Navid Mohammadi Foumani, Daniel F. Schmidt, and Geoffrey I. Webb

Monash University, Melbourne, Australia
angus.dempster@monash.edu

Abstract. Relatively little work in the field of time series classification focuses on learning effectively from very large quantities of data. Large datasets present significant practical challenges in terms of computational cost and memory complexity. We present strategies for extending two recent state-of-the-art methods for time series classification—namely, Hydra and Quant—to very large datasets. This allows for training these methods on large quantities of data with a fixed memory cost, while making effective use of appropriate computational resources. For Hydra, we fit a ridge regression classifier iteratively, using a single pass through the data, integrating the Hydra transform with the process of fitting the ridge regression model, allowing for a fixed memory cost, and allowing almost all computation to be performed on GPU. For Quant, we 'spread' subsets of extremely randomised trees over a given dataset such that each tree is trained using as much data as possible for a given amount of memory while minimising reads from the data, allowing for a simple tradeoff between error and computational cost. This allows for the straightforward application of both methods to very large quantities of data. We demonstrate these approaches with results (including learning curves) on a selection of large datasets with between approximately 85,000 and 47 million training examples.

Keywords: big data · hydra · quant · time series classification

1 Introduction

While some recent work in the field of time series classification focuses on computational efficiency [5,10,11], very little work actually deals with learning effectively from large quantities of data. 'State of the art' in the field of time series classification has come to mean state of the art in terms of accuracy on the datasets in the UCR archive [7,2,23]. Most of these datasets—at least, most of those which are commonly used for evaluation—are small. Median training set size for the 142 canonical univariate datasets is just 217 examples.

This stands in contrast to the datasets commonly used in other domains such as computer vision and natural language processing. It is not coincidental that, with some exceptions [18,17], deep learning methods have had a relatively muted impact on the field. These are generally low bias methods which require large quantities of training data. The field has also maintained a focus on an 'apples to apples' comparison of compute times based on a fixed amount of compute as a reference point—typically, a single CPU core [23]—potentially overlooking which methods are most efficient in practice, given available hardware (e.g., GPUs).

It may be that the 'bitter lesson'—i.e., that 'the only thing that matters in the long run is the leveraging of computation' [28]—has not yet been learned in time series classification. It may be that methods for time series classification which can exploit GPUs will be more useful in practice for large quantities of data, simply because they are amenable to being implemented in a way that suits available infrastructure per the 'hardware lottery' [15].

Many prominent methods for time series classification are limited by high computational complexity. In other words, these methods are computationally bound to relatively small datasets. However, training on very large quantities of data presents significant practical challenges, even for the most efficient methods, in terms of both memory complexity and computational cost.

Ultimately, memory does not scale with dataset size. It is impractical to require arbitrarily large memory in order to train a given model on increasingly large quantities of data. Further, methods which act on the entire dataset at once are not necessarily more efficient. For example, in practice, 'full' batch gradient descent is significantly less efficient than stochastic gradient descent.

To this end, we present strategies for extending two recent state-of-the-art methods (state of the art on the datasets in the UCR archive) for time series classification—namely, Hydra [10], and Quant [11]—to very large quantities of data. For Hydra, we train a ridge regression classifier iteratively, integrating the transform into the process of fitting ridge regression model. For Quant, we split the data into batches, training a subset of extremely randomised trees on each batch, ensuring that each tree is trained on as much data as possible within a given memory constraint.

The rest of this paper is structured as follows. Section 2 covers relevant related work. Section 3 sets out the strategies for training Hydra and Quant on large datasets. Section 4 presents experimental results, including learning curves, on select large datasets.

2 Background

The preeminence of the datasets in the UCR archive as a basis for benchmarking has meant that the field of time series classification has long been focused on smaller datasets. The field has not been seriously faced with the challenges and tradeoffs inherent in learning from very large quantities of data.

Many of the most prominent methods for time series classification have high computational complexity, requiring significant training time even for relatively small datasets [23]. However, even the most efficient methods face significant practical challenges in training on very large datasets in terms of computational cost and memory complexity.

Additionally, the bias–variance characteristics of methods currently considered state of the art in terms of classification error are likely optimised for smaller datasets, where error is minimised by reducing variance. For larger datasets—where variance decreases as dataset size increases—error is minimised by reducing bias [3].

Most of the datasets in the UCR archive are small. It is therefore reasonable to assume that methods considered state of the art on these datasets are effective at minimising variance. Methods for minimising variance include ensembling (e.g., 'hybrid' methods [22], and methods which use ensembles of decision trees), explicit regularisation (e.g., methods which use ℓ_2 or 'ridge' regularisation), and overparameterisation (e.g., the Rocket 'family' of methods [8], and other methods which combine a very large feature space with a linear classifier, such as WEASEL 2.0 [27]).

It is as yet unknown which, if any, current state-of-the-art methods are effective when trained on significantly larger quantities of data, or the extent to which these methods represent an appropriate balance of performance versus computational cost on larger datasets compared to, e.g., deep learning methods trained using GPUs.

2.1 Hydra

Hydra is a recent method for time series classification focused on computational efficiency. Hydra combines aspects of Rocket and dictionary methods [10]. Rocket transforms input time series using a large number of random convolutional kernels, and then uses the transformed features to train a ridge regression classifier [8]. Dictionary methods involve counting the occurrence of symbolic patterns in time series. Hydra combines the two approaches, counting the occurrence of random patterns—represented by random convolutional kernels—in the input time series.

Like other members of the Rocket 'family', Hydra uses a ridge regression classifier for smaller datasets. For larger datasets, it is intended to be used with logistic regression trained via stochastic gradient descent. In this setting, the time series in each batch are transformed, and the transformed features are used to perform an update step. This allows for a fixed memory cost (proportional to the size of the batch).

However, while reading and transforming the data in batches is memory efficient, computationally it is potentially very inefficient to repeatedly transform the same data over multiple epochs. In contrast to, e.g., a conventional convolutional neural network with learned weights, the convolutional kernels in Hydra (and other members of the Rocket 'family') are fixed. One of the main computational advantages of a fixed transform is being able to transform the data only once, i.e., to avoid the need to repeatedly transform the same data. If the transform cost is high (e.g., for long time series with a large number of channels), the time and computational cost spent repeatedly transforming the data over multiple epochs might well be better spent learning the weights of the convolutional kernels in a conventional convolutional neural network, rather than training a linear classifier on features produced by a fixed transform.

One approach is to store or cache the transformed features, so that each time series need only be transformed once [10]. For small datasets, it makes little difference whether the transformed features are stored or not (as the memory and computational costs are both small). As dataset size grows, caching the transformed features becomes more efficient. However, it becomes infeasible to store the transformed features for arbitrarily large data.

In this context, we present a strategy for training a ridge regression classifier iteratively with a single pass through the data, integrating the Hydra transform with the process of fitting the ridge regression model, obviating the need to store the transformed features.

2.2 Quant

Quant was recently found to be the fastest and most accurate interval method on the datasets in the newly-expanded UCR archive [23]. Interval methods involve computing summary statistics and miscellaneous other features over subseries (intervals) of the input time series [11]. Quant uses a single type of feature (i.e., quantiles), and an 'off the shelf' classifier, namely, extremely randomised trees. Quant has not previously been evaluated on datasets larger than those in the set of 142 datasets used in Middlehurst et al. [23].

There are various established approaches for training ensembles of decision trees on very large quantities of data, including training each tree on a subset of the training data ('pasting') [4], a subset of the features ('random subspaces'), or a subset of both examples and features ('random patches') [20,19].

However, most or all of this work assumes that the ensemble is being trained directly on the underlying data. Here, however, we need to both read and transform the data using the Quant transform, and then train the classifier on the transformed data. As such, we encounter a similar tradeoff between memory complexity and computational cost as for Hydra. It is untenable to cache or store the transformed features for arbitrarily large datasets. It is also computationally undesirable to repeatedly read and transform the same training examples, even if the Quant transform is relatively efficient. At the same time, as the results clearly show, it is desirable to train each tree on as much data as possible: see Section 4, below. To this end, we present an approach for training different subsets of trees on batches of data, where each batch is as large as possible subject to memory constraints.

3 Method

3.1 Hydra

The central challenge in extending Hydra (or any other member of the Rocket 'family' of methods) to very large datasets is resolving the tradeoff between memory complexity and computational cost. We train a ridge regression classifier iteratively, using a single pass through the data, integrating the Hydra transform into the process of fitting the ridge regression model. This allows for 'the best of both worlds' in the sense that we neither have to repeatedly read and transform the same data (each training example is read and transformed only once), or to store the transformed features (the transformed features are used to update an intermediate quantity used to fit the ridge regression model, and then discarded). Almost all computation can be performed on GPU. The same procedure can be used for any other member of the Rocket 'family' of methods (or any other fixed transform). However, Hydra has a significant advantage over other methods in this context, as discussed further below.

Fitting a ridge regression classifier involves fitting a ridge regression model where the classes have been encoded as regression targets, i.e., $\mathbf{Y} \in \{-1, +1\}$. Ridge regression uses a ridge or ℓ_2 penalty (given by λ), and has a closed-form solution [14, 64]:

$$\boldsymbol{\beta}_\lambda = (\mathbf{X}^\mathrm{T}\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\mathrm{T}\mathbf{Y}. \tag{1}$$

In the present context, $\mathbf{X}$ is an $n \times p$ feature matrix, representing the transformed features produced by Hydra, where n is the number of training examples, and p is the number of transformed features.

$\boldsymbol{p > n}$ In practical terms, for transforms such as Hydra, which produce a relatively small number of features, where $p > n$—i.e., where there are more features than training examples—this implies that the training set is relatively small. Accordingly, where $p > n$, we transform the entire training set, compute $\mathbf{X}\mathbf{X}^\mathrm{T}$, and use the 'shortcut' method (via eigendecomposition of $\mathbf{X}\mathbf{X}^\mathrm{T}$) in order to fit the ridge regression model and estimate LOOCV error for different candidate values of the ridge parameter. Computing $\mathbf{X}\mathbf{X}^\mathrm{T}$ reduces the $n \times p$ feature matrix to an $n \times n$ matrix. This makes the cost of fitting the ridge regression model proportional to n, regardless of the number of features, when $p > n$. The shortcut method has been discussed extensively elsewhere [30].

$\boldsymbol{n \geq p}$ With respect to (1), both $\mathbf{X}^\mathrm{T}\mathbf{X}$ and $\mathbf{X}^\mathrm{T}\mathbf{Y}$ can be computed iteratively. This follows from the definition of these two quantities:

$$\mathbf{X}^\mathrm{T}\mathbf{X} = \sum_i \mathbf{x}_i^\mathrm{T}\mathbf{x}_i \tag{2}$$

$$\mathbf{X}^\mathrm{T}\mathbf{Y} = \sum_i \mathbf{x}_i^\mathrm{T}\mathbf{y}_i. \tag{3}$$

Accordingly, where $n \geq p$, we read the data in batches, transform each batch using the Hydra transform, and then update $\mathbf{X}^\mathrm{T}\mathbf{X}$ and $\mathbf{X}^\mathrm{T}\mathbf{Y}$ per (2) and (3). The transformed features can then be discarded. Note also that the ordering of the data is irrelevant. $\mathbf{X}^\mathrm{T}\mathbf{X}$ and $\mathbf{X}^\mathrm{T}\mathbf{Y}$ are of size $p \times p$ and $p \times k$ respectively, where k is the number of classes. This makes the cost of fitting the ridge regression model proportional to p, regardless of n, where $n \geq p$.

While this approach is applicable to any member of the ROCKET 'family' of methods (or, indeed, any fixed transform), HYDRA has a significant advantage in this context. While ROCKET, MINIROCKET [9], and MULTIROCKET [29], for example, all use a fixed number of features (by default, between 10 and 50 thousand), HYDRA computes a number of features proportional to time series length. By default, HYDRA produces 512 features per dilation, where dilation is specified as $d \in 2^{\{0,1,\ldots,\lfloor \log_2 L \rfloor\}}$, where L is time series length. In other words, doubling the length of a time series only increases the number of features by 512. In practice, this means that except in the case of very long time series, HYDRA produces fewer features than the other transforms, and both $\mathbf{X}^{\mathrm{T}}\mathbf{X}$ and $\mathbf{X}^{\mathrm{T}}\mathbf{Y}$ ($p \times p$ and $p \times k$ respectively) are meaningfully smaller.

As n grows relative to p, using the shortcut method to compute estimated LOOCV error for different values of λ becomes increasingly less attractive, as it requires computing quantities which are proportional in size to n (fundamentally, it requires computing error per example). Accordingly, for $n \geq p$, we subsample a small validation set (being the smaller of 20% of the training set or 8,192 examples) prior to fitting the model. We choose the value of λ which minimises error on this validation set. (For both the 'shortcut' LOOCV procedure and the separate validation set we perform a grid search over 21 candidate values of λ across 12 orders of magnitude centred on $\sqrt{n}$.) In order to compute the inverse of $\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda\mathbf{I}$, we compute the eigendecomposition $\mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^{\mathrm{T}} = \mathbf{X}^{\mathrm{T}}\mathbf{X}$. The inverse is then given by: $(\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda\mathbf{I})^{-1} = \mathbf{V}(\mathbf{\Sigma}^2 + \lambda)^{-1}\mathbf{V}^{\mathrm{T}}$.

This procedure assumes that $\mathbf{X}$ is centred, and that we fit an intercept term. It is typical to normalise $\mathbf{X}$ by subtracting the mean and dividing by the standard deviation. As we are reading the data in batches, we compute the mean and standard deviation of (the columns of) $\mathbf{X}$, and the mean of $\mathbf{Y}$, as cumulative averages. This is very similar to batch norm [16]. We then compute:

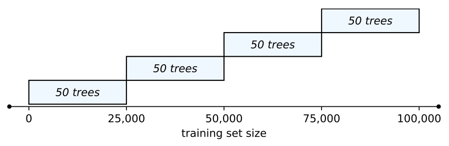
$$\mathbf{M}_{\mathbf{X}} = \boldsymbol{\mu}_{\mathbf{X}}^{\mathrm{T}}\boldsymbol{\mu}_{\mathbf{X}} \cdot n' \tag{4}$$

$$\boldsymbol{\Phi}_{\mathbf{X}} = \boldsymbol{\phi}_{\mathbf{X}}^{\mathrm{T}}\boldsymbol{\phi}_{\mathbf{X}} \tag{5}$$

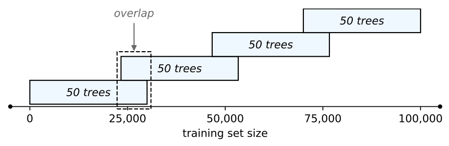
$$\mathbf{M}_{\mathbf{X},\mathbf{y}} = \boldsymbol{\mu}_{\mathbf{X}}^{\mathrm{T}}\boldsymbol{\mu}_{\mathbf{y}} \cdot n' \tag{6}$$

Here, $\boldsymbol{\mu}_{\mathbf{X}}$ and $\boldsymbol{\phi}_{\mathbf{X}}$ are row vectors representing the mean and standard deviation of (the columns of) $\mathbf{X}$, and n' is the size of the training set less the size of the validation set. (Accordingly, $\mathbf{M}_{\mathbf{X}}$ and $\boldsymbol{\Phi}_{\mathbf{X}}$ are matrices with the same shape as $\mathbf{X}^{\mathrm{T}}\mathbf{X}$, i.e., $p \times p$.) We then normalise $\mathbf{X}^{\mathrm{T}}\mathbf{X}$ by subtracting $\mathbf{M}_{\mathbf{X}}$ and dividing by $\boldsymbol{\Phi}_{\mathbf{X}}$ (elementwise), and normalise $\mathbf{X}^{\mathrm{T}}\mathbf{Y}$ by subtracting $\mathbf{M}_{\mathbf{X},\mathbf{y}}$ and dividing by $\boldsymbol{\phi}_{\mathbf{X}}$. The intercept term is $\boldsymbol{\mu}_{\mathbf{y}}$.

We note that this procedure results in the same model as a 'full' ridge regression fit, i.e., (1): it should produce the same model weights and predictions. However, our approach is both memory and compute efficient, where a 'full' fit is infeasible for very large quantities of data. Importantly, all of these operations—transforming the input time series, forming $\mathbf{X}^{\mathrm{T}}\mathbf{X}$ and $\mathbf{X}^{\mathrm{T}}\mathbf{Y}$, computing the eigendecomposition of $\mathbf{X}^{\mathrm{T}}\mathbf{X}$, fitting the model for different candidate values of λ and estimating error on the validation set—can be performed on GPU.



(a) Training set divisible by batch size.



(b) Training set not divisible by batch size.

Fig. 1. Batch size represents the maximum amount of data which can be read, transformed, and used to train trees for a given memory constraint. We divide the total number of trees between the batches and train a subset of trees on each batch. (If training set size is not divisible by batch size, the batches will overlap.) We shuffle the underlying training data such that each batch represents a random sample.

Multivariate Time Series Multivariate time series consist of time series with multiple channels, i.e., for a time series $\mathbf{Z} = (\mathbf{z}_0, \mathbf{z}_1, \ldots, \mathbf{z}_{L-1})$, each $\mathbf{z}_i$ is a vector of values, $\mathbf{z}_i = (z_{i,0}, z_{i,1}, z_{i,C-1})$ for C channels. For multivariate time series data, we use the original multivariate implementation of Hydra [10]. A different random subset of between 2 and 8 channels is assigned to each group of kernels. In effect, each group of kernels is applied to a univariate time series being the sum of a random subset of channels, with a different subset being used for each group. Following the distributive property of convolution, i.e., $X_0 * W + X_1 * W \equiv (X_0 + X_1) * W$, this is equivalent to applying the same kernel to each channel in the combination. This is conceptually similar to the multivariate implementation of MiniRocket [9].

3.2 Quant

For Quant, we propose reading and transforming the training set in batches, the batches being as large as possible for a given memory constraint, as illustrated in Figure 1. We divide the total number of trees between the batches and train a subset of trees on each batch. For example, for a total of 200 trees (the default for Quant), for four batches, we assign and train 50 trees on each batch. (The memory required per time series increases as the number of channels and time series length increase. For short time series with a small number of channels, this can result in reading very large batches, whereas for longer time series with many channels, this could result in smaller batches.) This ensures that: (a) each tree is trained on as much data as possible (subject to available memory); and (b) all of the training data is used at least once, while minimising the need to read and transform training examples. For smaller datasets, this defaults to simply training all trees on all of the training data. Where training set size is not divisible by batch size, the batches will overlap. However, the same training example will appear in at most two batches, so will be read and transformed at most twice. (This could be further optimised by only reading and transforming overlapping sections once.) We shuffle the data before forming the batches, such that batches smaller than the size of the training set represent approximately uniform random subsets.

This is effectively a simplified version of ‘pasting’ [4], where we minimise the number of times each training example is sampled, and maximise the amount of data used to train each tree. Louppe and Geurts [20,19] find that training decision trees on subsets of the training examples and/or subsets of features can produce similar classification error to training an ensemble of decision trees (e.g., extremely randomised trees) on the entire dataset. This is somewhat at odds with our results, which suggest that, on large datasets, error is a function of tree size, and that each tree should be trained on as much data as possible.

A closer examination of the findings in Louppe and Geurts [20] suggests some potential caveats. The actual differences in classification accuracy between different methods is small in absolute terms for most of the datasets evaluated. The largest training set used was comprised of only 35,000 training examples (with 784 features), i.e., the equivalent of only approximately 100 MB of training data (assuming float32). Most importantly, for the experiments on the three largest datasets, ensemble size was reduced and tree depth was limited, meaning that there is no real comparison to a model with trees trained to full depth on all the training data.

Pasting was originally proposed in a very different context in terms of the available computing resources. Nevertheless, the results in Breiman [4] in relation to pasting appear to be consistent with our results, i.e., that broadly speaking error is proportional to the quantity of data used to train each tree.

Our results clearly show that model complexity and 0–1 loss on large datasets are bounded by the quantity of data used to train each tree. There is a clear tradeoff between performance (0–1 loss) and computational and memory complexity. In terms of minimising 0–1 loss, it is clear that each tree should be trained on as much data as possible (within memory and computational constraints). This suggests that, ideally, all trees should be trained on all data, i.e., updating each tree using the data in each batch or, in other words, separating the question of memory complexity and batch size. We leave this for future work.

Multivariate Time Series The original implementation of Quant extends trivially to multivariate data, as the quantiles are already computed over each channel in a multivariate time series. The quantiles computed for each channel can be reshaped into a single set of features per time series (i.e., combining the quantiles from all channels). Indeed this appears to be how multivariate data is handled for the implementation of Quant available in the aeon toolkit [31]. Whether or not this is an optimal strategy for learning from multivariate data, both in terms of absolute performance (e.g., 0–1 loss), or computationally (in terms of the trade off between error and computational complexity), is untested.

4 Experiments

We demonstrate these approaches using five large datasets—*MosquitoSound*, *Pedestrian*, *S2Agri*, *Traffic*, and *WhaleSounds*—with a total of between approximately 105,000 and 59 million examples. In each case, we use 5-fold cross-validation (using predefined folds) such that, for each fold, approximately 80% of the data is used for training and 20% of the data is used for evaluation. Results presented here in terms of 0–1 loss and training time represent mean 0–1 loss and mean training time over the cross-validation folds.

We present results for two versions of Quant: (a) with a maximum batch size of 100 MB; and (b) with a maximum batch size of 1 GB. We compare the results for Quant and Hydra to results for DrCIF and HInceptionTime on four of the five datasets. We also present results for the datasets in the UEA multivariate time series classification archive [1].

Unless otherwise stated, Quant is trained using 4 CPU cores, and Hydra is trained using GPUs, on a cluster with Intel Xeon Gold CPUs and a mixture of NVIDIA V100, A40, and A100 GPUs (almost all jobs used V100 GPUs). Both methods are implemented in Python. Our code is available at: https://github.com/angus924/aaltd2024.

4.1 Datasets

MosquitoSound, taken from the broader UCR archive, consists of 279,566 (univariate) time series, each of length 3,750, representing recordings of wing-beats for six different species of mosquito [12]. The task is to identify the species of mosquito based on the recordings. The dataset has been split into stratified random cross-validation folds.

Pedestrian represents hourly pedestrian counts at various locations in Melbourne, Australia between 2009 and 2022 [6]. The processed dataset consists of 189,621 (univariate) time series, each of length 24. The task is to identify location based on the time series of counts. The dataset has been split into stratified random cross-validation folds.

S2Agri consists of pixel-level Sentinel-2 data at a 10m resolution [13,26]. The processed dataset contains 59,628,823 multivariate time series, with 10 channels (representing 10 spectral bands), each of length 24. This version of the dataset contains 34 classes representing different land cover types. The dataset has been split into cross-validation folds based on geographic location.

Traffic consists of hourly traffic counts at various locations in the state of NSW, Australia [32]. The processed dataset contains 1,460,968 (univariate) time series, each of length 24. The task is to predict the day of the week based on the time series of counts. The dataset has been split into stratified random cross-validation folds.

WhaleSounds consists of underwater acoustic recordings around Antarctica, manually annotated for seven different types of whale calls [25,24]. The dataset has been processed to extract the annotated whale calls from the original recordings. The processed dataset contains 105,163 (univariate) time series, each of length 2,500, with eight classes representing the seven types of whale call plus a class for unidentified/ambiguous sounds. This version of the dataset has been split into stratified random cross-validation folds.

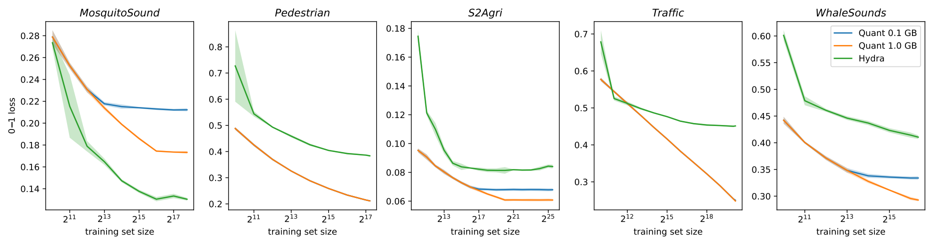


Fig. 2. Learning curves (0–1 loss) for QUANT (for batch sizes of 0.1 GB and 1.0 GB) and HYDRA on *MosquitoSound*, *Pedestrian*, *S2Agri*, *Traffic*, and *WhaleSounds*.

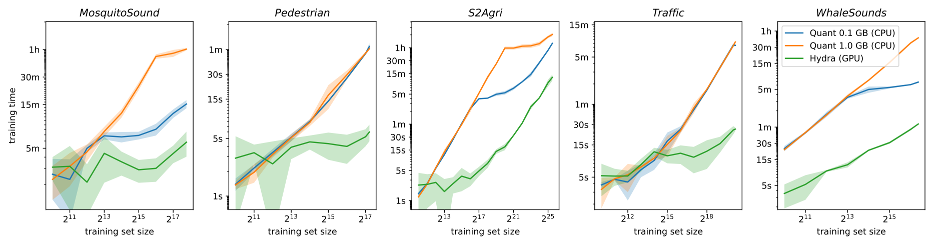


Fig. 3. Training times for QUANT (for batch sizes of 0.1 GB and 1.0 GB) and HYDRA on the same datasets.

4.2 Learning Curves

0–1 Loss Figure 2 shows learning curves (0–1 loss) for QUANT and HYDRA on *MosquitoSound*, *Pedestrian*, *S2Agri*, *Traffic*, and *WhaleSounds*. Figure 3 shows the corresponding training times for each method. Figure 2 shows that while HYDRA achieves lower 0–1 loss on *MosquitoSound*, QUANT achieves lower 0–1 loss on *Pedestrian*, *S2Agri*, *Traffic*, and *WhaleSounds*. (*MosquitoSound* is comprised of relatively long time series. The default hyperparameter settings for QUANT in terms of the number of intervals and the number of quantiles per interval may be suboptimal in this context.) *Pedestrian*, *S2Agri*, and *Traffic* are all similar in that they are comprised of relatively very short time series. We hesitate to draw firm conclusions in relation to the relative performance of the two methods on this sample of datasets.

Figure 2 shows that, for HYDRA, 0–1 loss continues to decrease (even if only modestly) even at the largest training set sizes, with the exception of *S2Agri*. (The small 'uptick' in error at the largest training set sizes for *S2Agri* appears to be due to a shortcoming in the process for determining the regularisation parameter. We leave the resolution of this point for future work.) We note that the combination of HYDRA and logistic regression may achieve lower 0–1 loss (at the expense of additional computational and/or memory complexity) as logistic regression is a more 'flexible' model, particularly where $n > p$. While both linear models, ridge regression is limited to models satisfying (1), whereas logistic regression is not. We leave a comparison of the two approaches to future work.

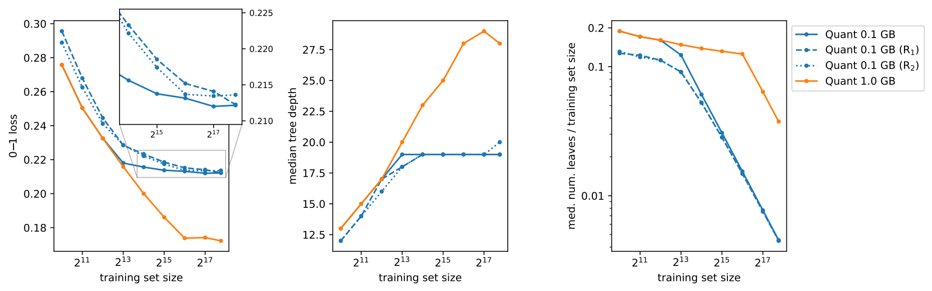


Fig. 4. 0–1 loss (left), median tree depth (centre), and the ratio of the median number of leaves per tree to training set size (right) for Quant (for maximum batch sizes of 0.1 GB and 1.0 GB) on *MosquitoSound*.

Figure 2 shows that, for Quant, 0–1 loss continues to decrease, even at the largest training set sizes, for *Pedestrian* and *Traffic*, but appears to largely stop decreasing, or decreases much more slowly after a certain point, for *MosquitoSound*, *S2Agri*, and *WhaleSounds*. This is strongly connected to the quantity of data used to train each tree (see further below).

Training Times Figure 3 shows the corresponding training times for each method. In one sense, the training times for Quant and Hydra are not comparable, as Quant is trained using CPUs, and Hydra is trained using GPUs. Ultimately, however, Hydra is able to make use of GPUs for training, whereas Quant is not (at least as currently implemented). As such, these results provide an indication of real-world training times for these methods on datasets of this size. At this scale, it is impractical to train different methods in such a way as to allow for a direct, 'apples to apples' comparison of training times, i.e., by training each method on a small fixed number of CPU cores.

Figure 3 shows that, for both Hydra and Quant, training time is essentially linear with training set size. Training times for Hydra are low in absolute terms, and relatively 'flat' on three of the datasets. On this point, training times for Hydra are affected by the relative size of n vs p (the computational cost of fitting the ridge regression model being proportional to $\min(n, p)$: see Section 3), the computational cost for the different validation schemes, and the relative expense of the transform versus the cost of fitting the ridge regression classifier.

For Quant, training time is dominated by the quantity of data used to train each tree. Once each batch reaches its maximum size (i.e., either 0.1 GB or 1.0 GB), the cost of training the ensemble on more data (but with the same maximum amount of data per tree) declines.

Model Complexity (Quant) Figure 4 shows 0–1 loss (left), median tree depth (centre), and the ratio of the median number of leaves per tree to training set size (right) for Quant, for maximum batch sizes of 0.1 GB and 1.0 GB, on

Table 1. 0–1 loss for QUANT, HYDRA, HInceptionTime, and DrCIF.

	QUANT$_{0.1}$	QUANT$_{1.0}$	HYDRA	HInception	DrCIF (4h)	DrCIF (8h)
MosquitoSound	0.2119	0.1731	0.1304	0.1743	0.2998	0.2394
Pedestrian	0.2115	0.2115	0.3856	0.3319	0.2228	0.2112
S2Agri	0.0675	0.0608	0.0845	—	—	—
Traffic	0.2508	0.2481	0.4508	0.3345	0.3667	0.3310
WhaleSounds	0.3339	0.2928	0.4092	0.4867	0.3522	0.3207

Table 2. Training times (excluding *S2Agri*).

QUANT$_{0.1}$	QUANT$_{1.0}$	HYDRA	HInception	DrCIF (4h)	DrCIF (8h)
32m 17s	1h 57m	8m 12s	6d	18h 25m	1d 9h

MosquitoSound (for a single fold). Figure 4 also shows results for two different methods for sampling batches with replacement (‘R_1’ and ‘R_2’).

Figure 4 shows very clearly that 0–1 loss is closely tied to model size, which is a function of the quantity of data used to train each tree. More data per tree allows for training deeper trees which, in turn, results in lower 0–1 loss. In other words, the quantity of data used to train each tree represents a kind of floor on 0–1 loss. While 0–1 loss continues to decrease modestly after reaching maximum batch size, there is a clear advantage to simply using more data.

The results are essentially the same when sampling batches with replacement (equivalent to bagging when training set size is smaller than maximum batch size and, in the limit, similar to sampling without replacement, to the extent that the probability of sampling the same object twice from a large set becomes increasingly small). Here we show two variants of sampling with replacement: using the same size and number of batches as the default, but sampling each batch with replacement (‘R_1’), and always sampling at least 50 batches (of up to maximum batch size, i.e., always training 4 trees per batch) and forming each batch with replacement (‘R_2’).

Figure 4 (right) shows that model size grows approximately linearly (training set size is approximately $5\times$ to $10\times$ the median number of leaves), until maximum batch size is reached, at which point model size stays essentially the same, while the quantity of data continues to increase. While the trees become relatively deep (with a median depth of approximately 28 by 2^{17} training examples), the actual model size (total number of nodes) is significantly smaller than a ‘full’ tree of the given depth, i.e., at 2^{17} training examples, for a median depth of 28, the median node count is just 16,740, i.e., closer to $\sqrt{2^{28}}$.

It seems clear that it would be advantageous to be able to continue training each tree on all data, subject to memory and computational constraints. We leave this for future work.

4.3 Comparisons with Other Methods

We present a comparison against preliminary results for DrCIF and HInceptionTime on four of the five datasets. (Training DrCIF and HInceptionTime

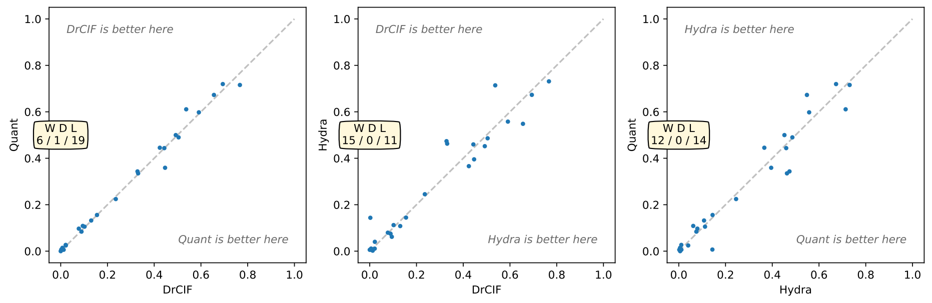


Fig. 5. Pairwise 0–1 loss for Quant vs DrCIF (left), Hydra vs DrCIF (centre), and Quant vs Hydra (right) on 26 datasets from the UEA multivariate time series classification archive.

on the *S2Agri* dataset is beyond the scope of this paper.) DrCIF is an interval method which achieves broadly similar results to Quant on the datasets in the UCR archive [21,22,23]. HInceptionTime is an ensemble of convolutional neural network models and represents the most accurate deep learning model on the datasets in the UCR archive [17,23].

Table 1 shows 0–1 loss for Quant and Hydra versus DrCIF and HInceptionTime for the full training set for four of the five datasets. With one exception, the results presented here represent mean results over five cross-validation folds. (Due to time and computational constraints, the results for HInceptionTime on *MosquitoSound* represent mean results over two folds.)

Table 1 shows that HInceptionTime achieves lower 0–1 loss than Hydra on two of the four datasets, but that Quant achieves lower 0–1 loss than HInceptionTime on all four datasets. DrCIF achieves lower 0–1 loss than Hydra on three of the four datasets, and lower 0–1 loss than Quant on one dataset.

Table 2 shows the total training time for each method (excluding *S2Agri*), averaged over the relevant folds. While these times are not directly comparable, it is clear that the outlier is HInceptionTime. HInceptionTime was trained using GPUs. DrCIF was trained using a single core per run using two different contract times: 4h and 8h. (Training runs for DrCIF did not reliably complete when using multiple threads/cores.) Error is clearly decreasing as DrCIF is allowed more time to train. Given the similarity between the results for DrCIF and Quant on the datasets in the UCR archive [23], and the UEA multivariate time series archive (see below), we expect that, with sufficient training time, 0–1 loss for DrCIF would at least match that for Quant on these datasets.

4.4 Multivariate Datasets

Results on the datasets in the UEA multivariate time series archive have not previously been presented for Hydra or Quant. Accordingly, we take the opportunity to present such results here, as they serve as a useful reference point for work involving multivariate time series data. Figure 5 shows pairwise 0–1 loss for QUANT vs DrCIF (left), HYDRA vs DrCIF (centre), and QUANT vs HYDRA (right). (Results for DrCIF are taken from Middlehurst et al. [22].) In relation to QUANT, given the relatively small size of these datasets, increasing the batch size beyond approximately 100 MB makes essentially no difference to the results (as almost all the datasets are smaller than 100 MB). As such, we show results for QUANT with a maximum batch size of 100 MB (this is essentially the default configuration). Figure 5 shows that, consistent with recent results on the expanded set of univariate datasets from the UCR archive [23], 0–1 loss for QUANT and DrCIF is very similar for most datasets. DrCIF achieves lower 0–1 loss than QUANT on 19 datasets, but the differences are mostly small. In contrast, 0–1 loss for HYDRA and DrCIF (and, by implication, QUANT) is less correlated. HYDRA achieves lower 0–1 loss than DrCIF on 15 datasets, although DrCIF achieves noticeably lower 0–1 loss on four of the datasets. Total training time (averaged over 30 folds) is 16 minutes 6 seconds for QUANT (CPU), and 1 minute 12 seconds for HYDRA (GPU).

5 Conclusion

The field of time series classification has long been focused on smaller datasets. Even in relation to more efficient methods, very little attention has been paid to learning effectively from very large quantities of data. Learning from large datasets requires making effective use of available computational resources, and operating in a context where dataset size might be considerably larger than available memory.

We present strategies for applying two state-of-the-art methods—HYDRA and QUANT—to large quantities of data. It is clear that in practical terms, HYDRA—trained using GPUs—is considerably faster than QUANT (by a large constant factor), although the difference could be reduced by using more CPU cores for QUANT where possible. However, QUANT achieves lower 0–1 loss than HYDRA on four of the five large datasets included in this study.

There are several important limitations to the strategies presented here. The efficient iterative method of fitting the ridge regression classifier is limited to linear (ridge) regression models. The additional flexibility of gradient descent (e.g., for training nonlinear models) is likely to allow for achieving lower 0–1 loss on larger datasets, albeit with additional computational cost and/or memory complexity. For QUANT, the results show that it is clearly desirable to train each tree with as much data as possible (subject to compute and memory constraints). The quantity of data used to train each tree forms a ceiling for model complexity and therefore the ability to learn from additional data. Nevertheless, we hope that the approaches set out here can help to form an efficient baseline for performance versus computational cost on larger datasets.

Acknowledgments. This work was supported by the Australian Research Council under award DP240100048.

References

1. Bagnall, A., Dau, H.A., Lines, J., Flynn, M., Large, J., Bostrom, A., Southam, P., Keogh, E.: The UEA multivariate time series classification archive (2018), arXiv:1811.00075
2. Bagnall, A., Lines, J., Bostrom, A., Large, J., Keogh, E.: The great time series classification bake off: A review and experimental evaluation of recent algorithmic advances. Data Mining and Knowledge Discovery **31**(3), 606–660 (2017)
3. Brain, D., Webb, G.I.: The need for low bias algorithms in classification learning from large data sets. In: Elomaa, T., Mannila, H., Toivonen, H. (eds.) Principles of Data Mining and Knowledge Discovery. pp. 62–73. Springer, Berlin (2002)
4. Breiman, L.: Pasting small votes for classification in large databases and on-line. Machine Learning **36**(1), 85–103 (1999)
5. Cabello, N., Naghizade, E., Qi, J., Kulik, L.: Fast, accurate and explainable time series classification through randomization. Data Mining and Knowledge Discovery (2023)
6. City of Melbourne: Pedestrian counting system. https://data.melbourne.vic.gov.au/explore/dataset/pedestrian-counting-system-monthly-counts-per-hour/information/ (2022), CC BY 4.0
7. Dau, H.A., Bagnall, A., Kamgar, K., Yeh, C.C.M., Zhu, Y., Gharghabi, S., Ratanamahatana, C.A., Keogh, E.: The UCR time series archive. IEEE/CAA Journal of Automatica Sinica **6**(6), 1293–1305 (2019)
8. Dempster, A., Petitjean, F., Webb, G.I.: Rocket: Exceptionally fast and accurate time series classification using random convolutional kernels. Data Mining and Knowledge Discovery **34**(5), 1454–1495 (2020)
9. Dempster, A., Schmidt, D.F., Webb, G.I.: 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. pp. 248–257. ACM, New York (2021)
10. Dempster, A., Schmidt, D.F., Webb, G.I.: Hydra: Competing convolutional kernels for fast and accurate time series classification. Data Mining and Knowledge Discovery (2023)
11. Dempster, A., Schmidt, D.F., Webb, G.I.: Quant: A minimalist interval method for time series classification. Data Mining and Knowledge Discovery (2024)
12. Fanioudakis, E., Geismar, M., Potamitis, I.: Mosquito wingbeat analysis and classification using deep learning. In: 26th European Signal Processing Conference. pp. 2410–2414 (2018)
13. Garnot, V.S.F., Landrieu, L., Giordano, S., Chehata, N.: Satellite image time series classification with pixel-set encoders and temporal self-attention (2020)
14. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer, NY (2009)
15. Hooker, S.: The hardware lottery. Communications of the ACM **64**(12), 58–65 (2021)
16. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: Proceedings of Machine Learning Research. pp. 448–456 (2015)
17. Ismail-Fawaz, A., Devanne, M., Weber, J., Forestier, G.: Deep learning for time series classification using new hand-crafted convolution filters. In: IEEE International Conference on Big Data. pp. 972–981 (2022)

18. Ismail Fawaz, H., Lucas, B., Forestier, G., Pelletier, C., Schmidt, D.F., Weber, J., Webb, G.I., Idoumghar, L., Muller, P., Petitjean, F.: InceptionTime: Finding AlexNet for time series classification. Data Mining and Knowledge Discovery **34**(6), 1936–1962 (2020)
19. Louppe, G.: Understanding Random Forests: From Theory to Practice. Ph.D. thesis, University of Liège (2014), arXiv:2305.11921
20. Louppe, G., Geurts, P.: Ensembles on random patches. In: Flach, P.A., De Bie, T., Cristianini, N. (eds.) Machine Learning and Knowledge Discovery in Databases. pp. 346–361. Springer, Berlin (2012)
21. Middlehurst, M., Large, J., Bagnall, A.: The Canonical Interval Forest (CIF) classifier for time series classification. In: IEEE International Conference on Big Data. pp. 188–195 (2020)
22. Middlehurst, M., Large, J., Flynn, M., Lines, J., Bostrom, A., Bagnall, A.: HIVE-COTE 2.0: A new meta ensemble for time series classification. Machine Learning **110**, 3211–3243 (2021b)
23. Middlehurst, M., Schäfer, P., Bagnall, A.: Bake off redux: A review and experimental evaluation of recent time series classification algorithms. Data Mining and Knowledge Discovery (2024)
24. Miller, B.S., Stafford, K.M., Van Opzeeland, I., Harris, D., Samaran, F., Širović, A., Buchan, S., Findlay, K., Balcazar, N., Nieukirk, S., Leroy, E.C., Aulich, M., Shabangu, F.W., Dziak, R.P., Lee, W.S., Hong, J.K.: An open access dataset for developing automated detectors of Antarctic baleen whale sounds and performance evaluation of two commonly used detectors. Scientific Reports **11** (2021)
25. Miller, B.S., Stafford, K.M., Van Opzeeland, I., et al.: Whale sounds. https://data.aad.gov.au/metadata/AcousticTrends_BlueFinLibrary (2020), CC BY 4.0
26. Sainte Fare Garnot, V., Landrieu, L.: S2Agri pixel set. https://zenodo.org/records/5815488 (2022), CC BY 4.0
27. Schäfer, P., Leser, U.: WEASEL 2.0: A random dilated dictionary transform for fast, accurate and memory constrained time series classification. Machine Learning **112**(12), 4763–4788 (2023)
28. Sutton, R.: The bitter lesson (2019), http://www.incompleteideas.net/IncIdeas/BitterLesson.html
29. Tan, C.W., Dempster, A., Bergmeir, C., Webb, G.I.: MultiRocket: Multiple pooling operators and transformations for fast and effective time series classification. Data Mining and Knowledge Discovery **36**(5), 1623–1646 (2022)
30. Tew, S., Boley, M., Schmidt, D.F.: Bayes beats cross validation: Efficient and accurate ridge regression via expectation maximization. In: 37th Conference on Neural Information Processing Systems (2023)
31. The aeon Developers: aeon. https://github.com/aeon-toolkit/aeon (2024)
32. Transport for NSW: NSW road traffic volume counts hourly. https://opendata.dev.transport.nsw.gov.au/dataset/nsw-roads-traffic-volume-counts-api/resource/bca06c7e-30be-4a90-bc8b-c67428c0823a (2023), CC BY 4.0