IMPERIAL

Department of Mathematics

Enhancing Gym Activity Recognition Through Hybrid Time Series Classification: Integrating Random Convolutional Kernels and Deep Learning Architectures

Christopher Adams

CID: 01059798

Supervised by Professor Almut Veraart

September 2024

Submitted in partial fulfilment of the requirements for the
MSc in Machine Learning and Data Science of
Imperial College London

The work contained in this thesis is my own work unless otherwise stated.

Signed: Christopher Adams Date: 2nd September 2024

Abstract

The proliferation of wearable smart devices and advancements in edge computing have transformed fitness activity tracking by enabling real-time monitoring and processing of human movement data. While a large body of research exists on Human Activity Recognition, Gym Activity Recognition (GAR) is a less explored and more complex research problem due to individual variations in exercise execution, exercises targeting the same muscle groups, as well as the vast number of exercises. Deployments using GAR classifiers can provide detailed exercise histories, allowing users to monitor progress towards their fitness goals.

Sensor selection and placement are crucial in GAR. This work references the MyoGym dataset, which uses a Myo Armband to capture six-dimensional motion data. By treating these data streams as multivariate time series, we apply Time Series Classification (TSC) methods, including the Rocket family of methods, which make use of random convolutional kernels, and deep learning architectures incorporating Gated Recurrent Units, which excel at modeling long-term dependencies.

We hypothesise that combining these distinct approaches into a hybrid model improves classification performance. Our research evaluates this hybrid classifier, focusing on its effectiveness compared to the individual standalone methods and on its ability to extract complementary features from the components.

Contents

1 Introduction

The advent of wearable smart devices and the raw movement data they capture has revolutionised the way fitness activities are tracked in the modern era. With advancements in edge computing, these smart devices have become increasingly ubiquitous, enabling real-time monitoring and processing of data streams, which in turn provides valuable feedback to users. A substantial body of research on Human Activity Recognition primarily focuses on classifying basic activities such as walking, cycling, and running, followed by their quantification. Gym Activity Recognition (GAR) is a significantly more complex research problem due to the unique way each individual performs exercises. Variations in repetition speed, inconsistent technique, and pauses between repetitions add to this complexity. Additionally, many gym exercises target the same muscle groups and have similar biomechanics, with subtle differences, such as performing an exercise on an incline, leading to distinct classifications. The range of gym exercises is also much broader, encompassing hundreds of different exercises compared to regular human activities. GAR classifiers can be designed and implemented to help users self monitor their fitness and evaluate their progress within the framework of their workout routines.

The selection and placement of sensors for GAR is a major design decision influenced by the research context. Recent methods have included placing a 3-axis accelerometer in a workout glove and another on the waist or chest to track posture (hao Chang et al. (2007)) and built-in accelerometers on smartphones directly placed on the weights (Pernek et al. (2013)). In this work, we reference Koskimäki et al. (2017) which introduces the MyoGym dataset used to train our methods. The Myo Armband, worn on the forearm, captures motion in six dimensions—three positional with an accelerometer and three rotational with a gyroscope. While classifiers like Random Forests or Logistic Regression could be directly applied to these data streams, we treat the data streams as multivariate time series and apply Time Series Classification (TSC) methods for classification.

This work presents a range of TSC methods applied to the MyoGym dataset, with a focus on approaches suitable for near-real-time classification. Of particular interest is the Rocket family of methods (Dempster et al. (2019)), which utilise random convolutional kernels to extract features from time series data. Additionally, we investigate modern deep learning architectures such as CNN-ResBiGRU (Mekruksavanich and Jitpattanakul (2024)) which incorporates Gated Recurrent Units designed to excel at modelling long term dependencies. These two methods demonstrated the strongest performance among those tested, yet their fundamentally different approaches led us to hypothesise that they provide complementary information. We propose that integrating them into a unified, multi-modal classification model could enhance overall classification performance. To test this hypothesis, we develop a hybrid classifier combining the outputs of both methods. Specifically, this research aims to address the following questions:

- Does the hybrid classifier outperform its individual components?

- Does the hybrid classifier effectively make use of features from both components, or does it render one redundant?
- How does fixing one component while training the other affect the performance of the hybrid model?

The rest of this work is structured as follows. In Chapter 2, we introduce the problem domain of TSC and review related work. In Chapter 3, we introduce the MyoGym dataset. We provide a commentary of the TSC methods in Chapter 4 and present the results of applying these methods on the MyoGym dataset in Chapter 5. The code used to implement the methods in this work can be found in the author's Github repository: Github

2 Background

In this chapter, we provide a short review of the existing body of literature of TSC methods. A notation scheme is then adopted and will be used consistently throughout this work. We formally define the problem of TSC and also examine foundational techniques, such as the Dynamic Time Warping metric, that are integral to many approaches in the literature.

2.1 Literature Review

Humanity's inherent desire is to want to learn from past events in order to anticipate the future. This eternal fact has inspired many forecasters over thousands of years to make guesses about the future and ultimately inspired the development of rigorous time series forecasting methods. This is not true of the domain of TSC, where only in recent years has most of the literature been developed, driven by the advent of more powerful computer processors which can handle the high demand of most TSC algorithms. In this chapter, we will introduce and critique a selection of these algorithms, dividing them into classical and neural network based methods.

Among the most ubiqutous classifiers in supervised learning is the Nearest Neighbour classifier in which the prediction for a new sample is based on the majority class of the nearest neighbours, calculated with a pre-chosen distance metric that calculates the dissimilarity of the sample and target. One obvious tool in the TSC toolbox is the Euclidean distance, yet it compares the values at each time point independently and will outright fail if the time series are of different lengths. These shortcomings motivated the development of the Dynamic Time Warping (DTW) metric introduced in Sakoe (1978) whose central idea is to wobble - or warp - the paths taken by a distance metric through the indexes of the sample and target time series. At each step of the path, either one or both path indexes are incremented, allowing the path to take different speeds through the sample and target time series, in such a way as to minimise the overall sum of distances along the path. This can be useful in real-world contexts when one time series is translated or sped up, something the Euclidean distance metric could not support. However, DTW has high algorithmic complexity due to the requirement to consider all possible paths through a sample and target time series and moreover allows for large time warps. To make these methods feasible, it has become standard practice to limit the time warps by imposing a constraint region that restricts the set of warping paths that it is possible to take. This decreases the computational cost as fewer possible paths need to be considered. The design of the constraint region is of course a hyperparameter, but popular methods include the Sakoe-Chiba band (Sakoe (1978)) and the Itakura parallelogram (Itakura (1975)), the latter allowing time warps to be larger in the middle of the time series and smaller close to the end points.

Discretisation of otherwise continuous valued time series is foundational to many other TSC methods. The most common discretisation approach is Symbolic Aggregation approXimation (SAX) (Lin et al. (2007)) where each observation is mapped to a corresponding bin. The edges of the bins may be computed using quantiles of the time series or uniformly based on the highest and lowest values. Next, techniques borrowed from natural language processing (NLP) can be used to extract informative features. The Bag of Patterns algorithm (Lin et al. (2012)) rolls a sliding window over the time series, extracts a word from the window and then computes a count of the occurrence of each word in the time series, analagous to a count vectoriser in NLP. Symbolic Aggregation approXimation in Vector Space Model (SAX-VSM) (Senin and Malinchik (2013)) computes a Term Frequency - Inverse Document Frequency (TF-IDF) matrix. Both methods incorporate short term dependencies which are preserved in the ordering of the letters in the words, but the long term dependencies are lost as the methods do not store the position in the time series a word was extracted from. This is a weakness partially addressed by the Symbolic Fourier Approximation (SFA) (Schäfer and Högqvist (2012)) which applies a Discrete Fourier Transform and chooses which frequencies to keep based on those with the highest F-statistics, then discretises, then concatenates the strengths of each kept frequency in each to form (ordered) words. The Bag-of-SFA-Symbols (BOSS) algorithm (Schäfer (2015)) implements SFA with sliding windows, treating each of the subsequences as a time series and following the SFA transformation. Analagously to NLP's count vectoriser, the frequency of each word in each time series is then extracted, so that the rows of the final matrix represent the time series, the columns represent the words extracted from the time series and the entries represent the word counts. The frequency domain also has its own TF-IDF analogue - Bag-of-SFA-Symbols in Vector Space (BOSS-VS) (Schäfer (2016)). Due to the Discrete Time Fourier Transform step, the granularity with which frequency based features can be extracted is a function of the window length, so methods like BOSS perform best with longer sliding windows. The drawback of discretisation more generally is that it reduces the ability to discern important but subtle differences between time series, so can trigger false classifications. This is especially pronounced in the problem instance of Human Activity Recognition, where noises or shocks to the recording device cause the measured time series to be incorrectly discretised, especially in the frequency domain where a shock at a single index can affect the measured strengths of multiple frequencies simultaneously. By choosing only the most dominant frequencies, SFA somewhat protects against this. Compensating for the loss of information during discretisation, the process itself is fast, and the final classification is typically performed with traditional machine learning classifiers like K-Nearest Neighbours or Random Forests.

Shapelets (Lines et al. (2012)) are snippets of consecutive time series observations that can be used to classify unlabelled time series. The distance between a shapelet and a time series can be defined as the minimum Euclidean distance between the shapelet and any of the subsequences of the time series of the same length as the shapelet; the most informative shapelets are extracted with the Shapelet Transform (Lines et al. (2012)) and used as feature extractors for conventional machine learning classifiers. The idea of Shapelets is intuitive, and the predictions are explainable by overlaying the shapelets onto the time series from which they were taken. By design, they capture local patterns which are more likely to be robust to noise and to be consistent within classes. Shapelets cannot, however, extract variable length sequences nor can they directly model longer term dependencies. Patterns that are distributed across classes are also likely to be missed. The biggest drawback is the computationally intensive nature of such a brute force method, something that Fast Shapelets (Keogh and Rakthanmanon (2013)) addresses by first applying SAX as described earlier, then projecting words to a smaller dimensionality by randomly masking some letters and trying to match exactly on the resulting hashes.

During a survey of popular TSC neural network architectures in the literature, Fawaz et al. (2018) found Convolutional Neural Network (CNN) architectures were the most widely applied. CNN's were originally developed for image processing, and Jiang et al. (2020) lay out an interesting pre-processing technique where a grayscale imagistic representation was first extracted from a one dimensional raw time series. A white pixel denoting the level of the time series at a time step was set against a black background. Fully Convolutional Networks Wang et al. (2016) were designed to classify univariate time series and do not contain any local pooling layers in order to maintain the length of the time series throughout the network. The final fully connected layer traditionally placed at the end of the architecture is also replaced with a Global Average Pooling layer. A popular architecture is the Residual Network (ResNet) (He et al. (2015)) whose defining property is the shortcut residual connection between the input of each block and the output of the convolutions inside it, addressing the vanishing and exploding gradient problems otherwise associated with long networks by providing a direct path for the back-propagation of gradients. ResNets are the deepest architecture reviewed and contain 9 convolutional layers followed by Global Average Pooling; it can learn complex hierarchical features from the time series data, but is computationally expensive and has longer inference times, posing challenges for real time classification. Arguably the most popular deep learning model for TSC is InceptionTime (Fawaz et al. (2019)) which is an ensemble comprising 5 Inception networks. Ensembling makes it competitive with the most accurate TSC currently in the literature, HIVE-COTE 2 (Middlehurst et al. (2021)), which it itself an ensemble of many classifiers, though is prohibitively expensive for most problem instances.

The Random Convolutional Kernel Transform (Rocket) (Dempster et al. (2019)) is the convolution of a large number of non-trainable kernels over the time series with different weights, dilations, biases, lengths and paddings which are randomly generated. In Rocket, 2 features are extracted from the application of each convolution; the global maximum and proportion of positive values. Then a classifier is trained on the outputs of the Rocket, selecting the most relevant features. The most commonly used classifer is the Ridge Classifier which can be used on data with a very large number of dimensions which may be higher even than the number of samples. Another is Adaboost (Freund and Schapire (1997)) which is a boosting techniques designed to learn from a large number of weak base learners. MiniRocket (Dempster et al. (2020)) was developed shortly thereafter, and uses pre-set kernels of length 9; in fact, MiniRocket is fully deterministic apart from sampling bias values and by drawing the bias using the entire dataset it can be made entirely deterministic. Because MiniRocket leverages a much smaller set of kernels, it requires less memory, scales to large datasets better and runs much faster, yet achieves comparable accuracy to Rocket. MultiRocket (Tan et al. (2021)) engineers additional features from the MiniRocket transform and applies the transform to both the raw time series and the first order differenced time series. Rocket architectures have low computational complexity and excellent performance compared to other modern TSC architectures, but use manually engineered features; while they have been applied with a single regression layer, they have not been applied in the context of larger networks.

Recurrent neural network (RNN) architectures were originally invented for time series analysis where sequence lengths can vary between dataset examples, making architectures that rely on the input having a fixed input size unsuitable. Long Short-Term

Memory networks were introduced by Hochreiter and Schmidhuber (1997) in order to address the vanishing gradient problem associated with backpropagating through a potentially long sequences of time steps. They mitigate the effect of vanishing gradients by integrating internal cell gates which control the propagation of information through time steps, allowing for the modelling of both long and short term dependencies. The GRU (Cho et al. (2014)) was released later, simplifying the forget and input gates in LSTM into a single update gate and so providing less fine-grained control over long term memory. In practice, these are implemented as layers in popular software libraries in the context of larger networks, and the chosen RNN is implemented bidirectionally, for example the approach taken in Mekruksavanich and Jitpattanakul (2024).

2.2 Introduction to Time Series Analysis

In this section, we outline the definitions and fundamental building blocks of time series analysis.

2.2.1 Time Series Definitions

This section provides some key definitions necessary for TSC.

Definition 2.2.1. Stochastic Process [1]

A **stochastic process** is a collection of random variables $\{X_t\}_{t \in T}$ defined on a common probability space $(\Omega, \mathcal{F}, P)$ and taking values in a state space $(S, \mathcal{S})$, where:

- Ω is the sample space,
- $\mathcal{F}$ is the σ-algebra on Ω,
- P is the probability measure,
- T is the index set
- S is the state space,
- $\mathcal{S}$ is the σ-algebra on S.

Let us assume that T is a set of time indexes, though in general T could also be used to denote spatial indexes. Then;

- If T is countable, then $\{X_t\}_{t \in T}$ is a discrete-time stochastic process.
- If T is uncountable, then $\{X_t\}_{t \in T}$ is a continuous-time stochastic process and can be denoted as $\{X(t)\}_{t \in T}$.

A time series represents the special case where T denotes the time set. In this work, we consider the case of discrete-time time series where $T \in \mathbb{Z}^+$. The values are context dependent and may be continuous or discrete.

Definition 2.2.2. Time Series

[1] Adapted from Chapter 7 of the MLDS Applicable Maths lecture notes, Autumn 2022

A set of observed values $x = \{x_{t_1}, ..., x_{t_T}\}$ with time indexes $\{t_1, ..., t_T\}$, where the length of the time series can be denoted by T.
Consider $x_{t_i} \in \mathbb{R}^{\mathbb{D}}$, $D \in \mathbb{Z}^+$.

- If $D = 1$, the time series is univariate.
- If $D > 1$, the time series is multivariate. Then x_{t_i} can be written as $\{x_{t_i,1}, ..., x_{t_i,d}, ..., x_{t_i,D}\}$.

Unless otherwise indicated, we will assume in this work that $\{t_1, ..., t_T\} = \{1, \ldots, T\}$.

2.2.2 Time Series Classification

In this section, we introduce the various domains within time series analysis and formally introduce the problem domain of TSC.

The task of forecasting requires making a prediction about the future value or values of a time series given a set of past observations, and a vast literature of techniques exist to tackle this problem. On the other hand, the demands TSC methods place on compute are so high that it has taken decades of advancements in compute availability to handle even moderately sized datasets. This is also true of time series clustering which is the unsupervised analagoue to TSC, which is to say the researcher does not have access to the time series labels.

TSC has a preponderance of real-life applications, notably in the healthcare sector where they are used to analyse electrocardiogram (ECG) and electroencephalogram (EEG) signals (Rushbrooke et al. (2023)) for diagnosing heart arrhythmias and detecting epileptic seizures. Another area where TSC models have proven effective is Human Activity Recognition, supported by the widespread use of camera and sensor technology, as well as the integration of real-time processing on smart devices (Yang et al. (2015)).

We now proceed to set up the definition of TSC.

Definition 2.2.3. Time Series Dataset

Suppose there is a collection of N time series, such that:

- x^n denotes the nth time series.
- y^n denotes the nth class label.
- C denotes the set of all possible class labels. Each $y^n \in C$.

Then a time series dataset is defined as:

$$\mathbb{D} = \{(x^1, y^1), \ldots, (x^n, y^n), \ldots, (x^N, y^N)\}, y^n \in C, n = 1, \ldots, N \tag{2.1}$$

We require for the purposes of this work that each time series have the same dimensionality, though they may have different lengths and time indexes.

Definition 2.2.4. Time Series Classification

TSC is a supervised learning task which involves assigning a single label y to a single time series x based on temporal dependencies and characteristics learned from the dataset $\mathbb{D}$. More formally, the goal of TSC is to learn a function $f : S^T \to C$ such that $f(x^i)$ predicts the correct class label y^i. We denote the class prediction made for our fitted classifier on time series x^i as $\hat{y}^i$.

2.2.3 Distances Between Time Series

Some TSC methods attempt to directly learn $f : S^T \to C$. Other TSC methods rely on the definition of the dissimilarity between a pair of time series called a mathematical distance metric. The distance metric is used to make a prediction for a new time series, normally using the majority class of the K - Nearest Neighbours in the dataset $\mathbb{D}$. The distance metric may follow from some transformation of the time series like discretisation or the Discrete Time Fourier Transform.

Suppose there are two time series $x = \{x_1, \ldots, x_{T_1}\}$, where $T_1 \in \mathbb{Z}^+$ and $y = \{y_1, \ldots, y_{T_2}\}$, where $T_2 \in \mathbb{Z}^+$.

The Euclidean distance is one of the simplest time series metrics. It is defined only when the lengths of the time series x and y are identical, i.e $T_1 = T_2 = T$.

Definition 2.2.5. Euclidean Distance

$$dist_{Euclidean}(x, y) = \sqrt{\Sigma_{i=1}^{T}(x_i - y_i)^2} \tag{2.2}$$

The Euclidean distance not being defined when x and y have different lengths forces us to compare the shorter time series with same length subsequences from the longer time series, truncating it and losing information that would help calculate the distance. Another drawback of the Euclidean distance is that it compares values at each time index independently, whereas the values of time series are correlated in time. In GAR, a repetition of an exercise may be performed slower or later compared to a second repetition. The Euclidean distance metric between the two repetitions would be large, even though the underlying signal is the same. The Dynamic Time Warping metric was developed to address this shortcoming and account for when the lengths of the time series x and y are different, i.e. $T_1 \neq T_2$. It is considered the baseline Nearest Neighbour metric for TSC. DTW (Sakoe (1978)) define a warping path of minimum total cost through pairs of time indexes from the two time series, with a carefully chosen cost function and path warping methodology.

The cost matrix $\mathcal{C}$ is the foundation on which DTW is computed. At an entry by entry level, it is the cost between pairs of values in the time series, so that $\mathcal{C}_{i,j} = f(x_i, y_j), i \in \{1, \ldots, T_1\}, j \in \{1, \ldots, T_2\}$. f is a function which evaluates the cost between any pair of time series values. It is a hyperparameter tailorable to a research question, but is commonly set to be the square of the Euclidean distance for multivariate time series.

Definition 2.2.6. Dynamic Time Warping Score (DTW Score)

Faouzi (2022) provides a tidy description of the construction of the warping path, defining a warping path of length L as a sequence $p = (p_1, \ldots, p_L)$ such that:

- Value Condition: $p_l = (i_l, j_l) \in \{1, \ldots, T_1\}$ x $\{1, \ldots, T_2\}, \forall l \in \{1, \ldots, L\}$.
- Boundary Condition: $p_1 = (1, 1)$, $p_L = (T_1, T_2)$
- Step Condition: $p_{l+1} - p_l \in \{(0, 1), (1, 0), (1, 1)\} \, \forall l \in \{1, \ldots, L - 1\}$

The total cost of a warping path p is therefore $\mathcal{C}_p(x, y) = \Sigma_{l=1}^{L} C_{i_l, j_l}$. Let $\mathbb{P}$ represent the set of warping paths. Then the DTW Score is:

$$DTW(x, y) = min_{p \in \mathbb{P}} \mathcal{C}_p(x, y) \tag{2.3}$$

Starting at the first time index in both time series, at each step in the path at least one of the time indexes is incremented, and the corresponding cost function computed. If exactly one time index is incremented, then the path can be said to have been warped. The cost of a warping path is the sum of costs at each step of the path. We also establish the definition of DTW as the path which has the smallest cost among all valid warping paths. Nevertheless, there are a large number of possible paths p for even moderately long time series. Enumerating and independently calculating the cost of each warping path is prohibitively expensive, but Sakoe (1978) also introduced a recurrence relation allowing us to use a dynamic programming approach.

Let $x_{.,i}$ and $y_{.,j}$ denote the sequence of time series indexes up to i and j respectively. Then $\forall i \geq 2$, $j \geq 2$:

$$DTW(x_{.,i}, y_{.,j}) = \mathcal{C}_{i,j} + min\{DTW(x_{.,i-1}, y_{.,j}), DTW(x_{.,i-1}, y_{.,j-1}), DTW(x_{.,i}, y_{.,j-1})\} \tag{2.4}$$

This holds true because of the step condition which prohibits incrementing either time index more than once in a single step. This recurrence relation hints at the requirement to store an underlying cumulative cost matrix. To get the DTW score for the two time series, simply read the (T_1, T_2)th element of that matrix.

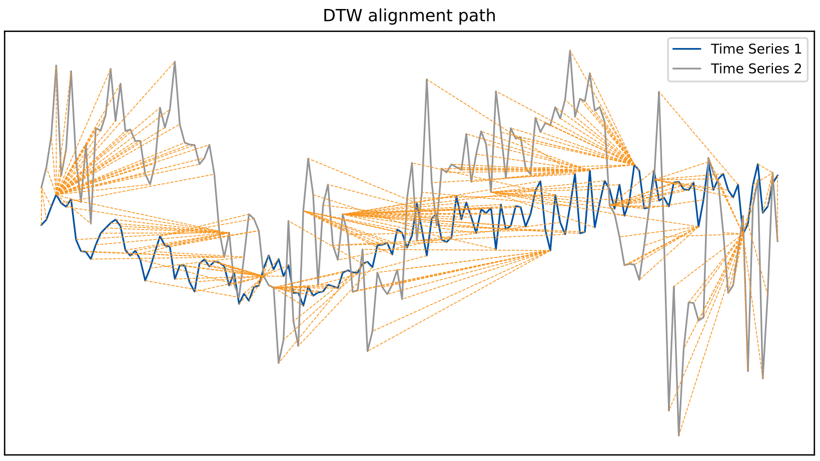


Figure 2.1: DTW alignment path between MyoGym train and test samples. Observe how a subset of time indexes in each sample are heavily leveraged by the DTW metric, matching with multiple time indexes in the other sample. This illustrates the warping process in action.

3 Preprocessing

In this chapter, we introduce the MyoGym dataset and discuss the pre-processing techniques used to prepare the dataset for the analysis described in the rest of this work.

3.1 MyoGym Dataset

The MyGym dataset was chosen for this work due to its public availability and recent widespread adoption for GAR tasks. The data were collected using a Myo Armband (Koskimäki et al. (2017)). Myo Armbands consist of 8 electromyogram (EMG) sensors and a 9-axis Inertial Measurement Unit containing a 3-axis gyroscope, a 3-axis accelerometer and a 3-axis magnetometer. The magnetometer data is not made available in the dataset. We discard the EMG sensor in this work as it requires hardware that is not available on a typical smart device, and the purpose of this work is to explore efficacy of classification approaches on data collected from devices that are ubiqutiously available. Therefore, we make use of only the accelerometer and gyroscope data, and we refer to each as sensor data where appropriate. The sensor data contains 6 dimensions which collectively describe the movement of the arm positionally and rotationally in a 3 dimensional space. The dataset was recorded with a conservative sampling rate of 50 Hz which is ideal to capture the frequencies that would be expected to be produced from gym activity data.

There are two labels and they are indexed to each observed value in a time series, not to the parent time series. The first label indexes the activity and the second indexes the trainer who performed the activity. The data were collected from 10 trainers as long bursts of continuous time series sampled at 50 Hz discrete intervals. Trainers completed sets of 10 repetitions from 30 different free weight based gym exercises. The exercises available were selected to cover every muscle group so as to introduce a meaningful inter-class variance, but the trainers were allowed to freely select the weights they used. There was also a null class used to represent data collected between distinct exercise sets in which the trainers moved around at the gym, swapped out the weights, stretched or just stayed still.

A list of the activities performed by the trainers is provided as a table in Appendix 7.1.

Remark 3.1.1. Accelerometers measure the acceleration of an object relative to an observer in free fall while gyroscopes measure the angular velocity. Assuming the starting position is known and the sensors sample with zero measurement error and at very high frequency, it is possible to recover the precise position and orientation of the object. This is known as **dead reckoning** and is a technique used in modern missile inertial guidance systems. It cannot be applied to recover a trajectory followed by a trainer completing an exercise repetition because small measurement errors in both sensors compound over time, in the case of accelerometer data, being integrated over twice to recover the position. This motivates our development of a classifier which is less sensitive to these measurement errors.

3.2 Exploratory Data Analysis

Figure 3.1 contains a plot for each axis in both the sensors. Notice that long periods of background inactivity are punctuated by shorter periods of activity. In fact, this is typical of MyoGym data which runs to approximately 11 hours of time series observations, of which 77% was considered null activity data. Periods of inactivity have high intra-class variance where the trainer appears to alternate between sharp movements and stillness, while periods of activity are clearly visible as short repetitions. The volume of null data eludes to a major class imbalance problem and it is excluded from Figure 3.2 where we count the number of observations in the dataset by activity.

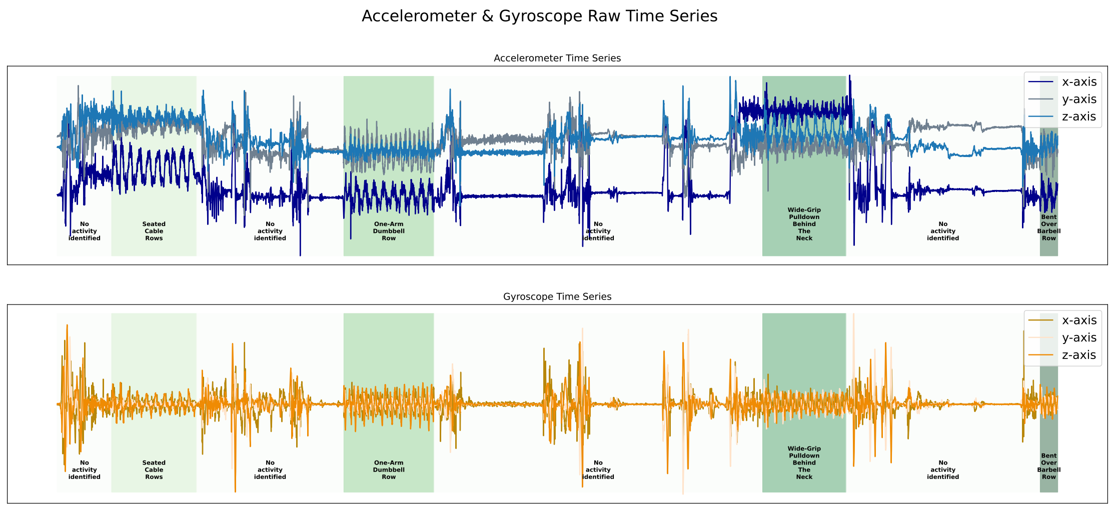


Figure 3.1: Long form time series of the sensor channels. Colour scheme taken from Mekruksavanich and Jitpattanakul (2024).

3.3 Data Cleansing

3.3.1 Data Normalisation

The orientation of the sensors during gym activity varies continuously which means that the effect of gravity on each direction and therefore the scale at which the measurements for that direction are recorded also varies continuously. On top of this, the accelerometer reacts suddenly to instantaneous shocks, whereas the effect of shocks is somewhat smoother in the velocity readings from the gyroscope. This causes some components of the sensor measurements to dominate other components, so we apply z-score normalisation. Not only does this reduce the effect of outliers like shocks, it also avoids some of the drawbacks of the other common normalisation technique, Min-Max scaling, which in the presence of outliers compacts the rest of the data to a narrower range, making training more tricky. Recognising each channel may have different attributes and properties, we apply z-score normalisation to each channel independently, rather than building a unified scaler.

Remark 3.3.1. GAR data is naturally noisy due to muscle tremors and measurement inaccuracies, either of which can introduce undesired frequencies into the signal. Unfortunately, MyoGym has been sampled at $f_s = 50$ Hz, meaning that the highest frequency that can be identified and filtered out is the Nyquist frequency $f_{max} = \frac{f_s}{2} = 25$ Hz.

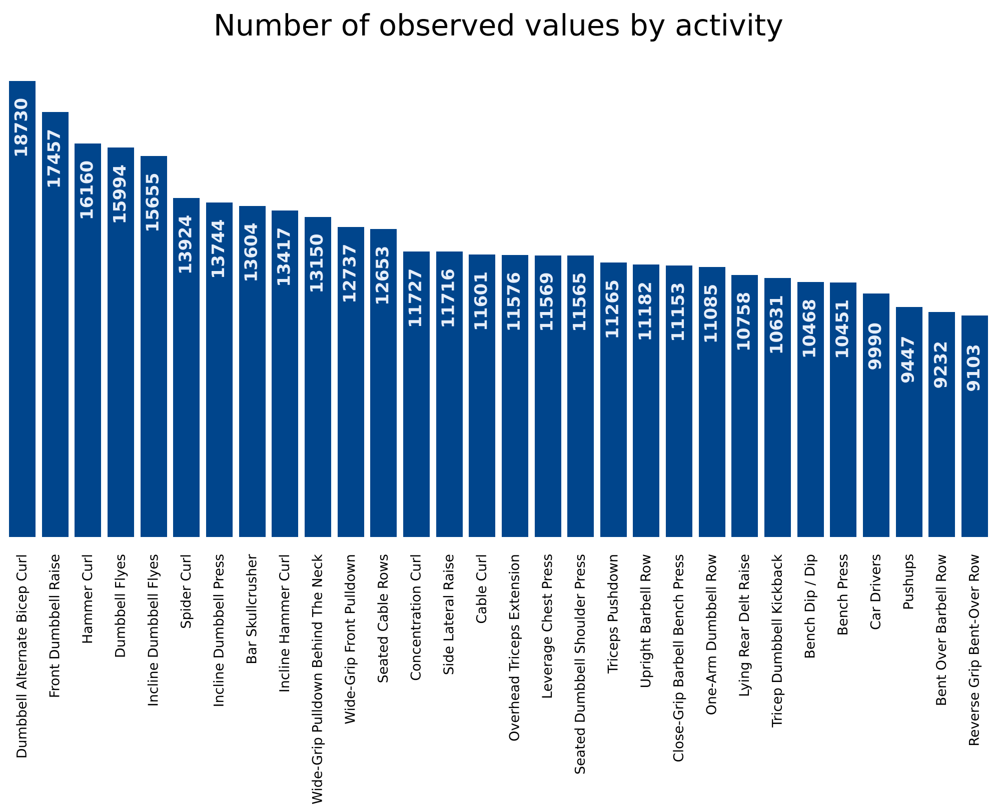


Figure 3.2: Counting the observed values by activity

This sample rate is probably too low to accurately reconstruct and eliminate noise frequencies. According to the Nyquist–Shannon sampling theorem, these frequencies will become aliased, meaning they will blend in with and be indistinguishable from lower frequencies in the signal. As a result, applying a low-pass filter would not be justifiable, as it cannot effectively differentiate between noise and useful signal components. Therefore, we cannot remove outliers this way, which would have been a necessary first step to applying a scaling approach like Min-Max scaling.

3.3.2 Data Segmentation

The MyoGym dataset includes extended time series from just 10 trainers during their workouts, which provides too few time series for training a conventional classifier. Moreover, since labels are assigned at the timestamp level and each time series contains multiple unique activities, the problem would require transformation. Otherwise, one would need to develop a multi-label classifier that can handle time series with evolving underlying signals.

To tackle this, we segment the data with the goal of generating a large number of shorter time series, each paired with its corresponding label. We achieve this by applying a 3 second (150 observations) sliding window over each time series, starting from the beginning. The window is then shifted forward by 1 second (50 observations) and reapplied repeatedly until the end of the time series is reached. This method results in consecutive windows overlapping by 2 seconds, while windows separated by one interval have a 1 second overlap. The transformed data produced from this process constitutes our dataset.

The window length and step size are critical hyperparameters that require careful consideration. Ideally, each window would perfectly encapsulate a single repetition, with its endpoints precisely matching the start and end of that repetition. However, trainers execute repetitions at varying speeds, and no labels are available to define the exact endpoints of these repetitions—a challenge recognized in the literature. Furthermore, during live testing, the beginning of a sliding window could coincide with any point within a repetition, creating uncertainty about where the repetition starts within the window. This uncertainty underscores the need for a sliding window approach. We set the window length to 3 seconds because this reflects the typical duration of a single repetition for most activities. Given that repetitions are intended to be identical, our assertion is that an additional repetition is unlikely to yield new information. The step size of 1 second was chosen to help the model learn from windows where the primary repetition that it captures starts at varying positions, enhancing robustness.

On a practical note, attentive readers may observe that windows capturing the moment a trainer switches activities—what we refer to as an activity changepoint—cannot be definitively labeled with a single activity. These windows are excluded from the dataset because handling such transitions is more effectively done through dedicated changepoint detection models rather than with TSC methods. The process is depicted in 3.3.

A selection of sliding windows compiled following the above method, extended to 500 observations (10 seconds) for readability, is provided in 3.4.

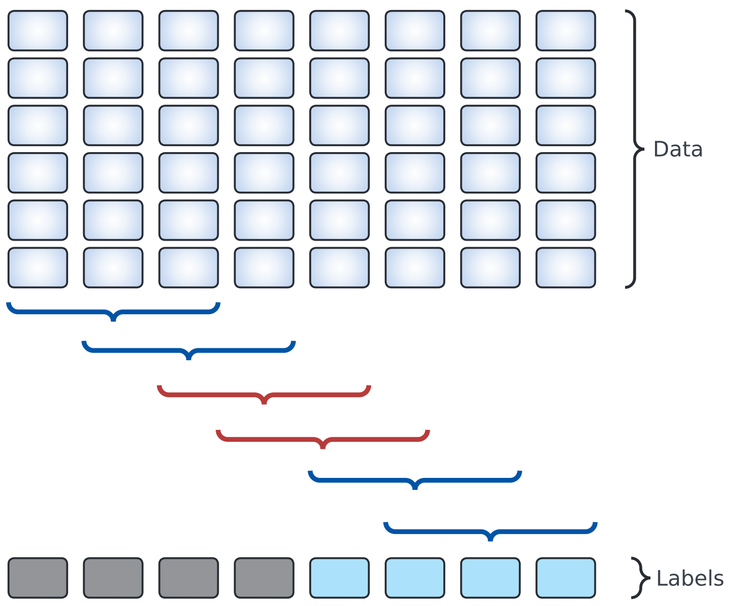


Figure 3.3: Data segmentation with sliding windows. Excluded windows containing activity changepoints are marked in red.

3.3.3 Data Splitting

In supervised machine learning tasks, it is generally assumed that the samples in the dataset are independently generated from an underlying function that the classifier aims to learn. Achieving good performance with most classification methods also requires training on a large number of samples, which we obtained by recycling time series data through sliding windows. This, however, violates the independence assumption, as each time step implicitly draws from prior steps. That said, because repetitions are cyclical, and trainers, with correct technique, consistently start and finish each repetition in the same physical position, it's reasonable to assert that sliding windows capturing different—though adjacent—repetitions will effectively be independent.

We split the dataset into train, validation and test sets. The train and validation sets were applied to 8 trainers; the test set is applied to the remaining 2 trainers who have been held out to test the true generalisation ability of the classifier.

3.3.4 Data Sampling

Recall from 3.2 that the class representing background activity comprised 77% of the MyoGym data, but from 3.2 that the remaining classes were sufficiently well balanced. As a final preprocessing step, the number of background activity samples in the train, validation and test sets is manually downsampled to be in line with the sample counts from the other activities.

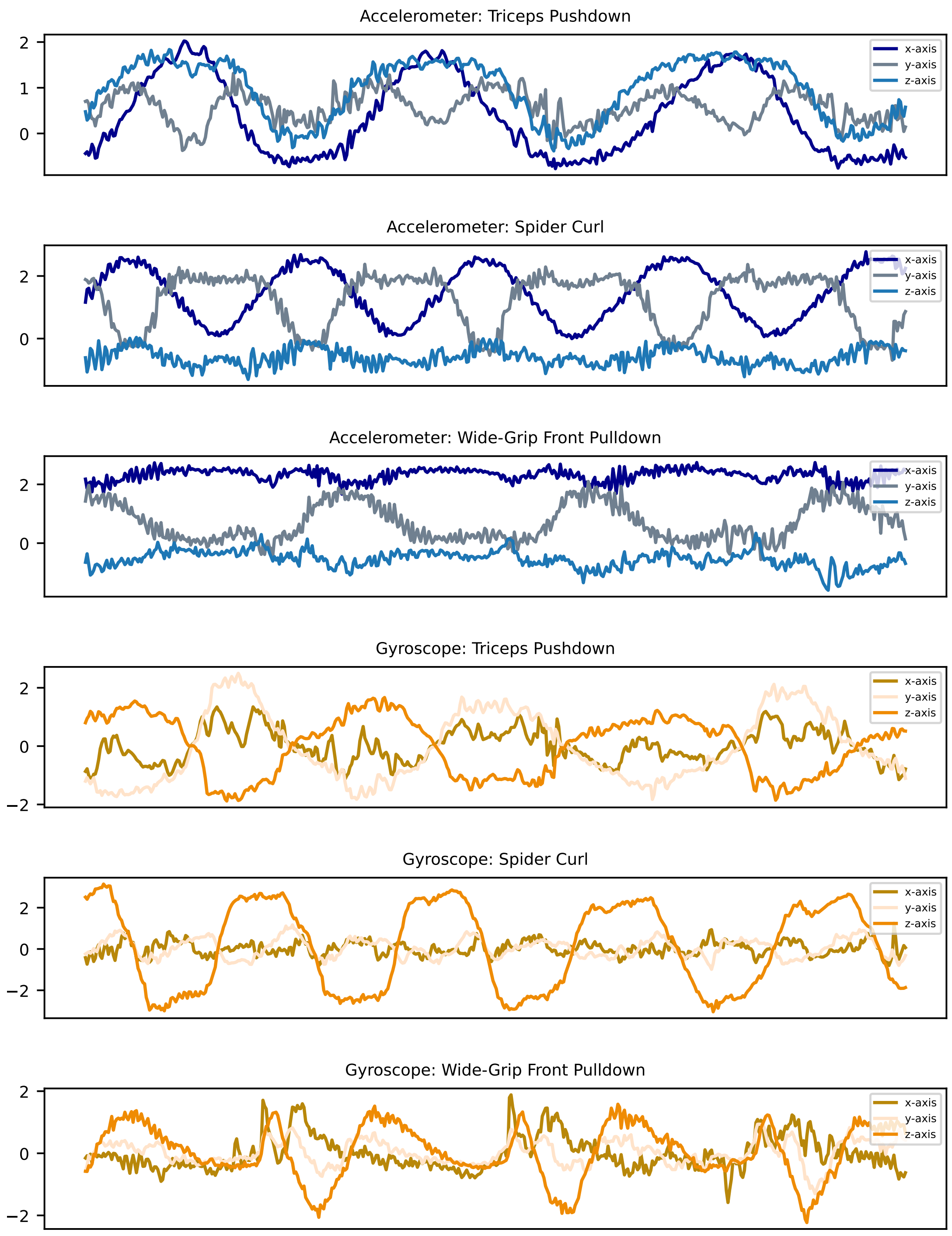


Figure 3.4: Samples of sensor data produced using sliding windows method. Note there is a clearly discernible underlying signal in each sample. Colour scheme taken from Mekruksavanich and Jitpattanakul (2024).

4 Methods

In this chapter, we describe a selection of TSC methods which will later be applied to the dataset introduced in chapter 3.

- Dynamic Time Warping & Nearest Neighbour Methods
- TSC Transformations & Classifiers: Shapelets, Discretisation, Random Convolutional Kernel Transform (Rocket)
- Deep Learning Methods: Convolutional & Recurrent Architectures
- **Hybrid Rocket: Hybrid Deep Learning & Rocket Architectures** (original contribution)

The motivation for testing a variety of methods lies in the difficulties of determining in advance which approach will most effectively learn the key features from the training data and apply that knowledge to accurately generalise to unseen examples.

4.1 DTW & Nearest Neighbour Methods

Among the most straightforward approaches to classify a time series sample is to take the majority class of the $K \in \mathbb{Z}^+$ least dissimilar examples whose classes the model has catalogued in advance, an approach referred to as K - Nearest Neighbours. DTW (2.2.6) is considered the default metric by which to judge the dissimilarity between two time series and is the classifier used in conjunction with the DTW based dissimilarity measures described below.

Brute Force Methods: It costs $O(T_1T_2)$ to compare a single pair of time series, i.e. the prediction cost for a full pairwise comparison between a prediction and train datasetof sizes N_{train} and N_{pred} respectively is $O(N_{train}N_{pred}T_1T_2)$. DTW itself is a metric largely closed to more efficient nearest neighbour search algorithms like KD-trees (Rau (2019)) because it does not satisfy the definitions of:

- **Distance**. It allows non-linear alignments, meaning it can stretch or compress x or y to find the best match. For a metric $d(x, y)$ to be a distance, $d(x, y) = 0 \iff x = y$.
- **Triangle inequality**. The combination of individual optimal alignments between time series x, y and z may not be the globally optimum path. The triangle inequality requires $d(x, z) \leq d(x, y) + d(y, z)$. KD-Trees rely on the triangle inequality to prune the search space.

The main drawback of the brute force approach is the prediction cost. However, the idea behind the barycenter method which is covered next is to choose a single highly representative element per class, reducing the prediction cost to $O(N_{pred}T_1T_2)$.

DTW Barycentre Averaging (DBA): DBA (Petitjean et al. (2011)) introduced a global averaging strategy which iteratively refines an average sequence in order to minimise the sum of squared DTW distances to the set of time series it is trying to represent. Initially, the DTW path between the average sequence and each time series from the set are computed. The pair of time indexes from both time series at each step in each DTW path are then *associated* with one another. Due to warping, this means a time index from one time series in the set may be associated with ≥ 1 time indexes in the average, and a time index in the average may be associated with ≥ 1 time indexes from each time series in the set. The total sum of squares is calculated by summing the partial sum of squares between each time index in the average and its associated time indexes. The partial sum of squares at time step in the average is minimised by computing the barycenter as the arithmetic mean of the associated time indexes. This is implemented as a two-step iterative process; the observant reader will observe the same approach is used to find barycenters in the popular K - Means clustering algorithm in unsupervised statistical learning:

- Step 1: Compute the DTW path between the current best estimate of the barycenter and each time series in the set. Label the associations.
- Step 2: Update each time index of the barycenter as the average of associated time indexes from step 1.

Soft-DTW Barycentre Averaging (SDBA): At each pair of time indexes (i, j) described in 2.4, we defined the costs and (implicitly) the alignment path recursively as $DTW(x_{.,i}, y_{.,j}) = \mathcal{C}_{i,j} + min\{DTW(x_{.,i-1}, y_{.,j}), DTW(x_{.,i-1}, y_{.,j-1}), DTW(x_{.,i}, y_{.,j-1})\}$. The minimum operation applied here is not differentiable, depriving us of the ability to minimise the DTW cost with gradient descent based methods. This eludes to a major drawback of DBA, and motivates Soft-DTW Barycenter Averaging (SDBA) (Cuturi and Blondel (2017)), which replaces the non-differentiable "min" operation with a differentiable "soft-min" operation (LogSumExp) - see 4.1. Soft-min is differentiable w.r.t. its inputs for $\gamma > 0$ so is a smooth approximation to the minimum.

$$softmin_\gamma(a_1, ..., a_n) = -\gamma log(\Sigma_i^n e^{-\frac{a_i}{\gamma}}) \tag{4.1}$$

The soft-min recursive updates become:

$$\phi_{i,j} = softmin\{SoftDTW(x_{.,i-1}, y_{.,j}), SoftDTW(x_{.,i-1}, y_{.,j-1}), SoftDTW(x_{.,i}, y_{.,j-1})\} \tag{4.2}$$

$$SoftDTW(x_{.,i}, y_{.,j}) = \mathcal{C}_{i,j} + \phi_{i,j} \tag{4.3}$$

The degree of smoothing in SDBA is determined by the parameter γ. As $\gamma \to 0$, the soft-min operation approaches the regular minimum function and allows us to recover classic DTW. SDBA is smoother and more easily differentiable for larger γ, and it is here that the method distinguishes itself apart from DBA.

4.2 Shapelets

A shapelet is a subsequence of consecutive observations from a time series which can be used to discriminate between activities. Denote a candidate shapelet by $s = (s_1, \ldots, s_L)$. The distance between the ith time series and the jth shapelet s^j is defined as:

$$d(x,s) = min_{k \in \{1,\ldots,T-L+1\}} \Sigma_{l=1}^{L} (s_l - x_{k+l})^2 \tag{4.4}$$

$$M_{i,j} = d(x^i, s^j) \tag{4.5}$$

The 2D distance matrix $M \in N$ x K is introduced in 4.5 to store the distances in 4.4. The key step is the selection of the *best* K shapelets from which M is constructed, where K is a hyperparameter to be determined by the user. We consider one method to extract these best shapelets from a dataset and a second method to learn the best shapelets via gradient descent.

Extracting Best Shapelets: Jason et al. (2015) introduce the Shapelet Transform which considered only shapelets which can be found in some x^i in the dataset. It explicitly calculates M after which we can apply any standard machine learning classifier. The disassociation between these two steps allows us to consider different classification approaches and merge other time series features into a classification pipeline. In this work, we found the K - Nearest Neighbours classifier to work best.

For each x^i in the dataset, all possible shapelets of lengths $l \in [length_{min}, length_{max}]$ are extracted and their distance 4.4 to x^i is cached. $length_{min}$ and $length_{max}$ are hyperparameters and are incumbent on the user to carefully select. Then, the shapelets are sorted in order of *quality*. Shapelets with overlapping indices are termed self-similar shapelets and are removed as they do not provide truly complementary information. The final step involves merging the newly extracted shapelets with the current best K shapelets, replacing any that are outperformed by our quality measure, while discarding the remaining new shapelets to keep the cumulative list within the top K.

While the quality metric for ranking shapelets can be based on information gain, this work follows Jason et al. (2015) and instead uses the F-statistic to measure the difference of means in an Analysis of Variance (ANOVA). The groups are defined by the columns of M, and the F-statistic evaluates the ratio of variance between these groups to the variance within them. Shapelets with the highest F-statistics are considered the most informative and are thus ranked as the best shapelets.

Learning Best Shapelets: Shapelet learning (Grabocka et al. (2014)) directly learns optimal shapelets by defining a differentiable classification model and updating it with stochastic gradient descent, forgoing the searching process. We describe the formulation of the objective function for a binary classifier and use shapelets of fixed length L. Extending to $\mathcal{C}$ targets follows a similar process but would require the formulation of $\mathcal{C}$ one vs all binary classification problems.

$$\hat{y}^i = \sigma(w_0 + \Sigma_{k=1}^{K} M_{i,j} w_j) \tag{4.6}$$

$$L(y^i, \hat{y}^i) = -y^i ln(\hat{y}^i) - (1 - y^i) ln(1 - \hat{y}^i) \tag{4.7}$$

We use the model provided in 4.6 to estimate $\hat{y}^i$, where σ denotes the standard logistic function. We learn the weights $w = (w_0, \dots, w_K)$ by applying stochastic gradient descent on the standard Binary Cross Entropy (BCE) loss function for sample i provided in 4.7 (the BCE is in practice regularised with $L2$ penalisation). This is therefore a joint optimisation problem of the loss given in 4.7 over all the shapelets and weights w.

The main contribution of the SDBA method was to define a differentiable soft minimum. Jason et al. (2015) also presents a differentiable analague to the minimum operation of 4.4. Define D in 4.9 where the (i, j, k)th index stores the distance between time series i and shapelet j at time indexes $[k, k + L - 1]$, $k \in [1, T - K + 1]$. Then 4.10 defines the smooth soft minimum, which approaches 4.5 as $\alpha \to -\infty$.

$$d^*(x, s, k) = \frac{1}{L}\Sigma_{l=1}^{L}(x_{k+l-1} - s_l)^2 \tag{4.8}$$

$$D_{i,j,k} = d^*(x^i, s^j, k) \tag{4.9}$$

$$\tilde{M}_{i,j} = \frac{\Sigma_{k=1}^{T-L+1} D_{i,j,k} e^{\alpha D_{i,j,k}}}{\Sigma_{k'=1}^{T-L+1} e^{\alpha D_{i,j,k'}}} \tag{4.10}$$

Notice that the only variable inputs to the soft minimum function are the shapelets themselves. 4.11 provides the full gradient equation w.r.t point l in the jth shapelet. A breakdown of the differentiated terms is easy to derive, but the interested reader can refer to Jason et al. (2015). The 1st term is obtained by differentiation of 4.7, the 2nd term from 4.6 and the 3rd and 4th terms in the chained component in 4.10 and 4.8 & 4.9 respectively.

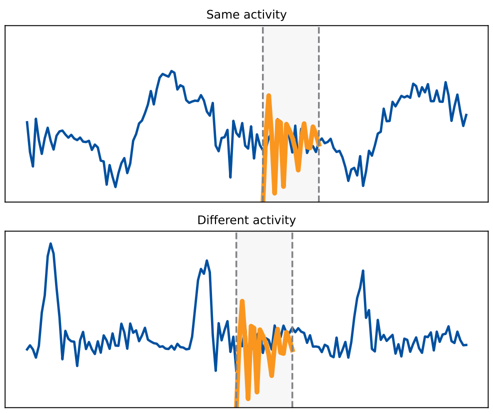
$$\frac{\partial L(y^i, \hat{y}^i)}{s_l^j} = \frac{\partial L(y^i, \hat{y}^i)}{\partial \hat{y}^i} \frac{\partial \hat{y}^i}{\partial \tilde{M}_{i,j}} \Sigma_{k=1}^{T-L+1} \frac{\partial \tilde{M}_{i,j}}{\partial D_{i,j,k}} \frac{\partial D_{i,j,k}}{\partial s_l^j} \tag{4.11}$$

One immutable advantage to learning classifiers is the ability to learn shapelets which are highly discriminative of the classes while not necessarily being part of the dataset. Learning shapelets is also less expensive than the Shapelet Transform, costing $O(NT^2)$ per iteration. Unlike the Shapelet Transform however, the transformation and classifier must both be optimised together when learning shapelets.

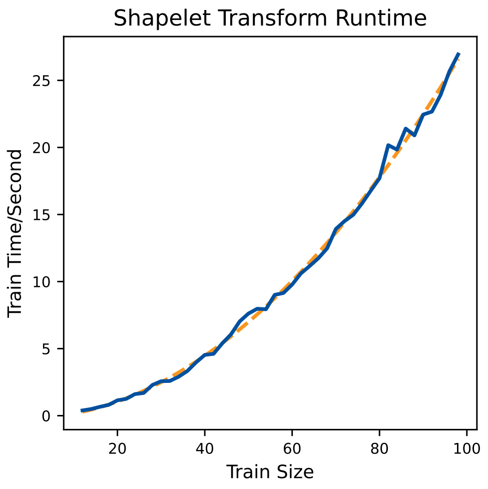
4.3 Discretisation

Discretising time series invites a delicate tradeoff between preserving the information within a time series and transforming a TSC problem to one that allows different methods traditionally employed for Natural Language Processing (NLP) to be used. Symbolic Aggregation approXimation (SAX) (Lin et al. (2007)) is the most common approach to discretisation. Each observation is mapped to a bin. The edges of the bins can be computed using quantiles of the time series or uniformly based on the highest and lowest values.

The output of SAX is a sequence of discretised values for each time series, which we will refer to as letters. The Bag of Patterns algorithm (Lin et al. (2012)) slides a window over the time series, concatenating letters in the window into words before counting the occurrence of each word in the time series. The MyoGym data is multidimensional, so we instead apply SAX to each channel, then concatenate over the channels to construct



(a) Sample shapelet overlaid on two time series: one from the same activity and another from a different activity.



(b) Shapelet Transform runtime increases quadaratically with train data size. Dashed orange line denotes a fitted curve.

Figure 4.1

words. This allows us to incorporate multi-dimensionality at the expense of modelling temporal relationships. The other method we implement is Symbolic Aggregation approXimation in Vector Space Model (SAX-VSM) (Senin and Malinchik (2013)). If Bag of Patterns is analagous to a Count Vectoriser in NLP, then SAX-VSM is analagous to a Term Frequency - Inverse Document Frequency (TF-IDF) matrix; it weights words according to how many other time series they can be found in.

4.4 Random Convolutional Kernel Transform (Rocket)

4.4.1 Time Series Convolutions

The convolution is a foundational operation used in problem domains like band pass filtering (as we neglected to perform in 3), image processing (blurring, sharpening and edge detection can be considered special cases of convolutions), the Rocket Transformation and Convolutional Neural Networks (CNNs). We denote the convolution operation between a time series x and a filter kernel w by $*$ so that the output is $y = x * w$.

Definition 4.4.1. Time Series Convolution

$$y_t = [x * w]_t = \Sigma_{k=-\infty}^{\infty} x_{t-k+1} w_k \tag{4.12}$$

In this work, w is of length $K \approx 10$ so has finite support $[1, \ldots, K]$. We refer to $[w_1, \ldots, w_K]$ as the weights. We can therefore rewrite the convolution operation as in 4.4.2. If x is multivariate, the kernel it is convolved with has an additional dimension and the convolution has an additional summand over the variables.

Definition 4.4.2. Time Series Convolution (Finite Support)

$$y_t = [x * w]_t = \Sigma_{k=1}^{K} x_{t-k+1} w_k \tag{4.13}$$

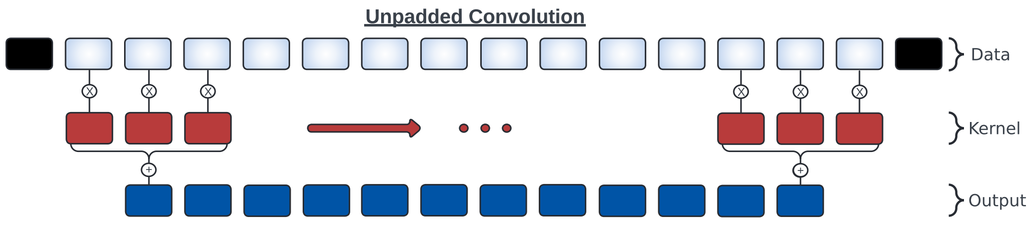


Figure 4.2: No padding.

Padding: Figure 4.3 shows that padding the original time series on both ends with zero values extends the support of x beyond $[1, \dots, T]$. This means we can define the summation in 4.4.2 over a longer input signal with the benefit of maintaining the original support in the output series y. If no padding is used, then y has length $[1, \dots, T-K+1]$ as it must be defined from summands 4.4.2 where $x \in [1, \dots, T]$.

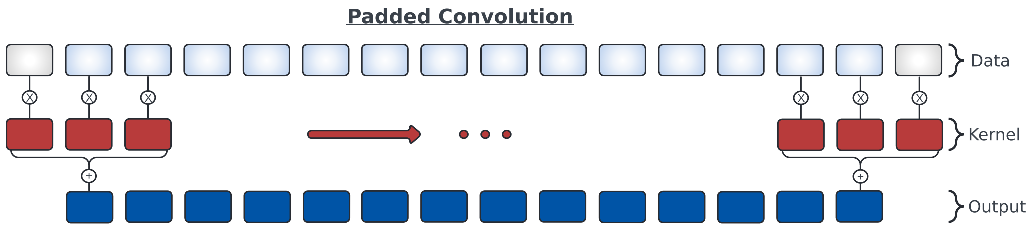


Figure 4.3: *Same* padding. The output time series is of the same length as the input time series due to zero-padding.

Dilation: Convolutions by design only incorporate the region of the input that a kernel is elementwise multiplied over. This is the receptive field; we may desire a larger receptive field if wanting to capture more global context. We can make it larger by using larger kernels or by stacking several kernels, but both require large numbers of weights. Dilations are designed to increase the size of the receptive field of the filter without increasing the number of weights. Figure 4.4 represents a dilation; it is the insertion of gaps between the elements of the filter, where the gap size is known as the dilation rate, so the filter can cover a wider area of the input and capture more global context. If the dilation rate is d, the kth element of the filter corresponds to the input time index $t - (k-1)d$.

4.4.2 Rocket

CNN architectures typically comprise many convolutional layers to extract features at different resolutions. Training the convolutional kernels during stochastic gradient descent is very expensive, often requiring many epochs to converge. The idea of the Random Convolutional Kernel Transform (Dempster et al. (2019)) is to generate and then apply a large number P (often, $P \approx 10,000$) of random non-trainable kernels using different values of the: kernel length, weights, bias, dilation and padding. A small number Q of features are then extracted from the output from applying each kernel to

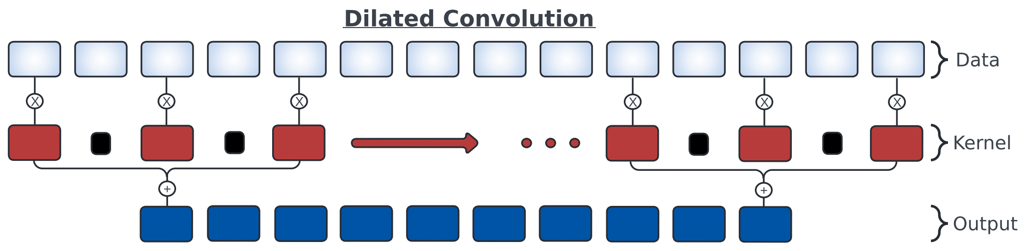


Figure 4.4: Dilation with no padding. Black components of the kernel are zero entries. The output time series is shorter than the input time series, but the resolution is the same.

conclude the transformation. In fact, the output of a Rocket transform has dimension N x (PQ). In the context of MyoGym, $PQ >> N$ which allows a subsequent classification stage to perfectly memorise the training data including any noise; this is modelled at the expense of the ability to generalise to unseen data. Therefore, Rocket was originally implemented with a Ridge Classifier which selects only the most informative features for classification, penalising other weights.

The Rocket Transformation combined with Ridge Classification has much lower cost than a trainable architecture of a similar size. The Minimally Random Convolutional Kernel Transform (MiniRocket) Dempster et al. (2020) was proposed as an extension of Rocket and has a yet lower cost. It achieves this by fixing the properties of the kernel rather than sampling from a distribution. We adapt the table of comparison provided in Dempster et al. (2020) to better compare both methodologies.

Kernel Property	**Rocket**	**MiniRocket**
Length	Random length $\in \{7, 9, 11\}$	Length 9
Weights	$w_i \sim \mathcal{N}(0,\, 1), i = 1, \dots, K$ [1]	{-1, 2}
Number of unique kernels	10,000	$\binom{9}{3} = 84$ [2]
Bias	$b \sim \mathcal{U}(-1, 1)$	From convolution output
Dilation	Random	Fixed
Padding	$f(x) = \begin{cases} \text{None}, & 0.5 \\ \text{Same}, & 0.5 \end{cases}$	Alternating
Extracted features	Global Max Pooling + PPV	PPV

MiniRocket has the advantage of being almost fully deterministic; the properties of the kernels are pre-selected to have a much smaller cardinality, making MiniRocket a much more efficient transform to implement in practice. The only stochastic element of MiniRocket is choosing the train sample from which to compute the bias values. The bias values are then taken from the quantiles of the convolution output - this part is deterministic. By drawing the bias values from the entire training set rather than from a randomly selected sample, MiniRocket can be made fully deterministic.

[1] Rocket then centers the weights with $w_i = w_i - \bar{w}$, where $\bar{w}$ is the weight mean average.

[2] MiniRocket uses the subset of kernels of length 9 with 3 values of +2 and 6 values of -1, for a total of $\binom{9}{3}$ kernels. Without constaining the values in this way, there are 9! possible kernels.

It was mentioned earlier that Q features are extracted from each convolutional output. Rocket ($Q = 2$) computes the Proportion of Positive Values (PPV) and Global Max Pooling (MaxPool). MiniRocket ($Q = 1$) computes PPV. PPV is time invariant to two shifted time series with the same pattern, which suits our dataset because the beginning of a sliding window could coincide with any point within a repetition, but each repetition should be consistent. In contrast, MaxPool is robust to noise, capturing the most extreme values in the time series that might be caused by strong spikes. Both PPV and Max are also linear time operations.

By adding an extra dimension to the convolutional kernel, Rocket based methods are readily extendable to multivariate time series. To extract PPV and MaxPool, there is an additional summand over the time series channel.

4.5 Deep Learning Methods

Deep Learning has been extensively applied in TSC, though much of the literature recycle architectures or techniques originally developed for image processing. We give a detailed description of a selection of existing architectures found to transfer well to our problem instance.

Fully Convolutional Network (FCN): Wang et al. (2016) propose applying Fully Connected Networks as a strong baseline classifier. The FCN comprises a sequence of convolutional blocks, interspersed with batch normalisations and Rectified Linear Unit (ReLU) activation functions to scale the convolution outputs. FCNs are so-called because they comprise no pooling or fully connected layers, preserving the information contained in each time index from the original input through the learned non-linear transformations in each block, ultimately allowing for time-index level classification. Different kernel sizes {8, 5, 3} are used in each block to capture features at different scales and each block increments the size of the receptive field in relation to the kernel size. We seek a single class per time series, so repurpose the FCN by following it up with global max pooling and a fully connected layer with $\mathcal{C}$ classes (see Figure 4.5).

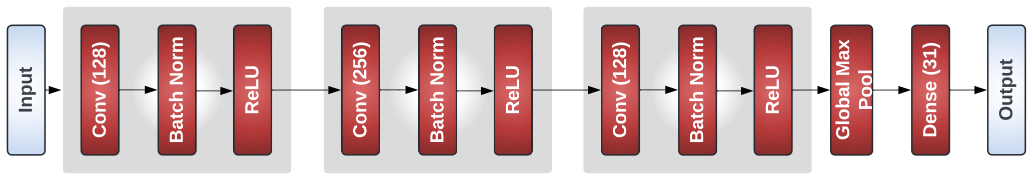


Figure 4.5: FCN Architecture. There are 128, 256 and 128 filters in the 3 convolutional blocks respectively.

InceptionTime: Proposed by Fawaz et al. (2019), InceptionTime is a popular ensemble based on the famed Inception Network (Szegedy et al. (2014)) but adapted for TSC. While state of the art ensemble architectures like Hierarchical Vote Collective of Transformation-based Ensembles 2 (HIVE-COTE 2) (Middlehurst et al. (2021)) are a so-called *meta-ensemble* of classifiers like the Shapelet Transform and the Rocket Transform (among other methods), they are extremely expensive TSC approaches and are wholly unsuitable for near real-time GAR tasks. On the contrary, while each Inception Network has randomly initialised weights, the networks themselves are architecturally identical.

Each Inception Network can be divided into residual blocks, concluding with a global averaging pooling layer and a fully connected layer with $\mathcal{C}$ classes for classification. Each residual block can be further divided into Inception Modules, which are run in sequence with shortcut linear connections between the input and output of the block. [1] The purpose of the residual connection is to tackle the vanishing gradient problem where the weights in early layers in a large network receive extremely small updates. Residual connections provide a direct path for the backpropagation of gradients, imitating a shallower network by allowing either of the blocks to be bypassed if they do not improve the training performance, in which case that block would learn an identity mapping.

The principle contribution of the Inception network on which InceptionTime was based was the Inception module, which begins by processing the input time series of dimension M through a bottleneck convolutional layer with k length 1 convolutional filters. We call this a bottleneck because $k << M$. It forces the Inception module to learn a compact representation of the time series, reducing the number of parameters to be learned. Next, in order to learn features at multiple scales, convolutions of lengths {10, 20, 40} are simultaneously slided over the output of the bottleneck layer. In parallel to all this, the input time series is pooled to introduce some invariance to time and then a second bottleneck is applied, which is concatenated to the outputs of the multiple simultaneous convolutions. This is the final Inception module output, a building block for larger architectures.

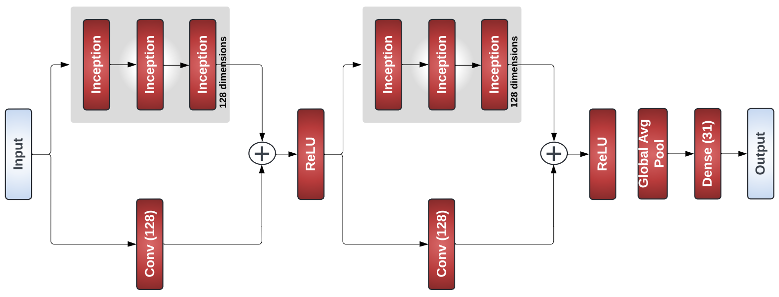


Figure 4.6: InceptionTime. The architecture we implement features 2 residual blocks.

CNN-ResBiGRU: Mekruksavanich and Jitpattanakul (2024) introduce a deep learning model centered around an architectural building block customised for GAR which they call the ResBiGRU. It is repeated several times, with the first ResBiGRU block being preceded by multiple convolutional layers and the final ResBiGRU block being succeeded by global max pooling and a fully connected layer with $\mathcal{C}$ classes to make the

[1]The input and output of the block must have the same length and dimensions in order for the addition of the shortcut connection to be mathematically valid. We ensure the lengths match by padding. We ensure the dimensions match by applying a length 1 kernel with the required output dimensionality to the input. This technique is called a projection shortcut connection.

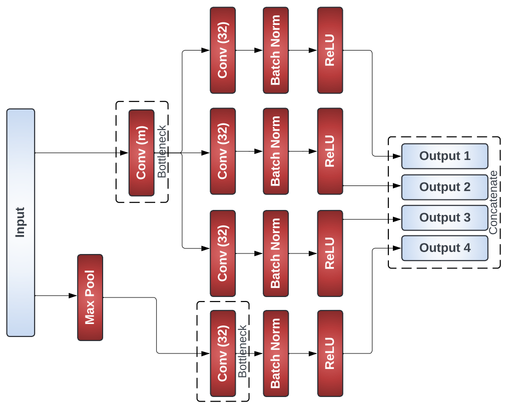


Figure 4.7: Inception Module. Each convolution comprises 32 kernels for a total of 32 x 4 = 128 output channels. Batch normalisation and a ReLU activation function are typically applied to the outputs of each individual kernel before concatenation.

classification. Residual Bidirectional Gated Recurrent Units (ResBiGRUs) themselves integrate a residual connection (discussed earlier in InceptionTime) with a bidirectional [2] GRU, which is a type of Recurrent Neural Network (RNN) first published in Cho et al. (2014). RNNs take advantage of the structure of time series by maintaining a memory of previous inputs, enabling them to better extract patterns in temporal dependencies.

The main idea of an RNN is the recurrent unit which can be considered a non-linear mapping from $\{x_i \in \mathbb{R}^D, i \leq t\}$ to an output $\hat{y}_t \in \mathbb{R}^O$. There is also a hidden layer $h_t \in \mathbb{R}^H$ which recursively defines each h_t from h_{t-1} and the observation x_t. The update equations for both the hidden and output layers are provided in figures 4.14 and 4.15 below. The weights W_{xh}, W_{hh} and W_{hy} and the corresponding biases b_h and b_o are all learnable parameters, while σ almost generally denotes the sigmoid function.

$$h_t = \sigma(W_{xh}^T x_t + W_{hh}h_{t-1} + b_h), W_{xh} \in \mathbb{R}^{D \text{ x } H}, w_{hh} \in \mathbb{R}^{H \text{ x } H}, b_h \in \mathbb{R}^H \tag{4.14}$$

$$\hat{y}_t = \sigma(W_{hy}^T h_t + b_o), W_{hy} \in \mathbb{R}^{H \text{ x } O}, b_o \in \mathbb{R}^O \tag{4.15}$$

Weight sharing enables RNNs to compactly store temporal relationships. Because backpropagation calculations are performed backwards *through time*, the vanishing gradient problem makes it difficult for recurrent units to learn long-range temporal dependencies for long time series such as MyoGym. However, GRUs incorporate gating mechanisms which, similar to residual shortcut connections, retain information and propagate gradients over longer time horizons. Specifically, GRUs incorporate a reset gate vector $r_t \in \mathbb{R}^R$ and an update gate vector $z_t \in \mathbb{R}^Z$. In TSC GRUs are most frequently used as feature extractors and so it has become notational convention that the output of the

[2] A regular GRU (usually) processes a time series in the forward direction, meaning it can only capture dependencies of the current time index on past time indexes. In a bidirectional GRU, a second GRU is applied to the reverse direction, which captures additional information held in the dependencies of the current time index on future time indexes. This is appropriate in contexts like TSC for MyoGym where all time steps are known.

GRU $\hat{y}_t = h_t$.

$$z_t = \sigma(W_{xz}^T x_t + W_{hz}^T h_{t-1} + b_z), W_{xz} \in \mathbb{R}^{D \text{ x } Z}, W_{hz} \in \mathbb{R}^{H \text{ x } Z}, b_z \in \mathbb{R}^Z \tag{4.16}$$

$$r_t = \sigma(W_{xr}^T x_t + W_{hr}^T h_{t-1} + b_r), W_{xr} \in \mathbb{R}^{D \text{ x } R}, w_{hr} \in \mathbb{R}^{H \text{ x } R}, b_r \in \mathbb{R}^R \tag{4.17}$$

$$h_t = (1-z_t) \circ h_{t-1} + z_t \circ \tanh(W_{xh}^T x_t + W_{hh}^T (r_t \circ h_{t-1}) + b_h), W_{xh} \in \mathbb{R}^{D \text{ x } H}, w_{hh} \in \mathbb{R}^{H \text{ x } H}, b_h \in \mathbb{R}^H \tag{4.18}$$

$\circ$ denotes the elementwise Hadamard product. The σ activation forces $z_t, r_t \in [0, 1]$, meaning z_t can properly act as the weights in the hidden state equation 4.18. z_t's role is to determine how much of the previous hidden state should be retained, so 4.18 has an elegant (though not precisely correct) interpretation: $h_t = P(\text{Keep } h_{t-1})h_{t-1} + P(\text{Discard } h_{t-1})\text{Recurrent Unit}(x_t, h_{t-1})$. The recurrent unit analagy is not exactly correct because of the presence of the extra reset gate, whose role is to selectively remember and forget different components of h_{t-1}, providing additional flexibility to the calculation beyond a simple componentwise weighted average.

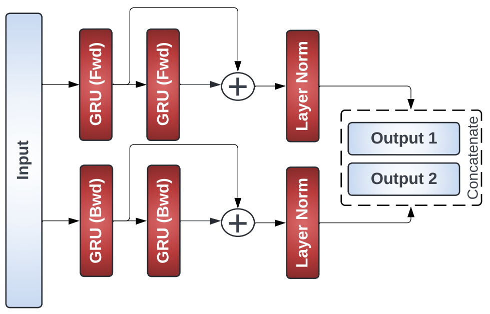


Figure 4.8: The ResBiGRU block introduced by Mekruksavanich and Jitpattanakul (2024). The GRU is applied twice in sequence and the results of each application are added together. GRU is also applied bidirectionally, and the outputs from the forward and backward direction are concatenated.

4.6 Hybrid Rocket: Hybrid Deep Learning & Rocket Architectures

In the domain of GAR, there is a premium on TSC approaches with low compute cost to support near-real time classification on smart devices. The proposed Hybrid Rocket methods seek to integrate the diverse feature space generated by Rocket-based techniques with the learned weights from the existing CNN-ResBiGRU network.

MiniRocket's computational simplicity made it the fastest tested method. A more interesting property of Rocket architectures is their reliance on randomly generating a very large number of kernels, enriching the feature space. Using random kernels also ensures the model is not constrained by design to favouring the identification of certain types of dependencies. Whereas, CNN-ResBiGRU represents a traditional neural network, trained through parameter optimisation to reduce a loss function. By replacing the Ridge Classifier usually employed after MiniRocket with a hybrid network integration, we aim to determine if the random feature space from MiniRocket improves the robustness of the hybrid model relative to the CNN-ResBiGRU on its own.

As discussed earlier in the commentary on Rocket methods, having $PQ >> N$ enables the training of a perfect classifier on the training data by optimisating PQ x C weights. However, the large number of weights tends to cause the classifier to overfit and memorise the training data, increasing the risk of getting stuck in local minima. Given that Ridge Classification has proven effective in guiding a classifier to focus on the most relevant features, we propose Hybrid Rocket models that integrate L2 regularisation in the fully connected layer immediately following the MiniRocket transform.

L2 regularisation aims to reduce the effective number of weights within the *same* layer by heavily penalising the magnitudes of the least informative components. Additionally, we explore dimensionality reduction techniques to define a smaller number $m << PQ$ of dimensions that still capture most of the variance from the original MiniRocket transform, thereby reducing the effective number of weights in *subsequent* layers. We use Principle Components Analysis (PCA) for this purpose, which identifies and retains the m most significant principle components. The number m is chosen empirically as the elbow point, where the variance explained by subsequent additional components sharply decreases. We expect each MiniRocket feature to have broadly the same informativeness, and we observe empirically in Figure 4.9, that the elbow point is $m \approx 128$, which we use in subsequent modelling.

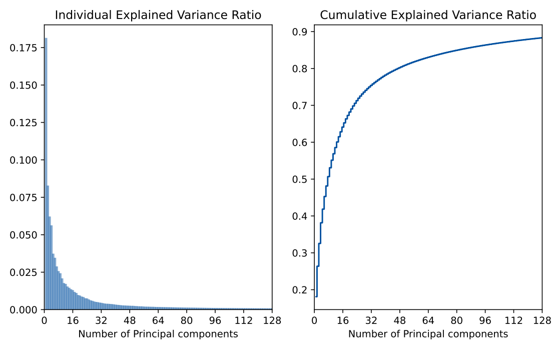


Figure 4.9: The most informative principle components. The top 128 principle components retain 88.3% of the variance of the original MiniRocket transform.

Training Hybrid Rocket amounts to a joint optimisation over multimodal inputs: one time series and one bag of MiniRocket features. Below, we break down a few empirical challenges in multimodal backpropagation and the techniques used to navigate them.

- **Input Scaling**: Each input should be on the same scale. The CNN-ResBiGRU component is layer normalised in the ResBiGRU block and the PPV columns in the MiniRocket features are in $[0, 1]$. All the hybrid models use batch normalisation after the output of both components.

- **Incorporating Both Components**: We experiment with an alternating training approach where one of the components is trained while the other component is held constant. This is intended to encourage both components to learn, rather than rely only on the component with dominant gradient updates.
- **Learning Rates**: In general, the most effective learning rate for each component will be different. We invoke separate optimisers in the component-wise training model and employ callbacks to reduce the learning rates of the optimisers when the validation loss plateaus.

Bringing this all together, we consider the various permutations of hybrid models that arise from L2 regularisation of the MiniRocket transform versus PCA and a full versus component-wise training approach (the latter also incorporating separate optimisers).

Proposed Model	**L2 / PCA**	**Full/Component-wise**
Full Hybrid Rocket	L2	Full
Full Hybrid PCA	PCA	Full
Component Hybrid Rocket	L2	Component-wisel
Component Hybrid PCA	PCA	Component-wisel

Below, we provide the pseudocode for the component-wise training approach: notice the weights combining both components are never fixed.

Algorithm 1 Component-wise Hybrid Rocket training

```
Inputs:
    Time series dataset: $\mathbb{D}$
    MiniRocket output: $\mathcal{R}$
    Number of epochs: epochs

Initialise weights:
Self contained in CNNResBiGRU component: $w_{\mathbb{D}}$
Self contained in MiniRocket component: $w_{\mathcal{R}}$
Combining CNNResBiGRU and MiniRocket components: $w_{\mathcal{C}}$

for epoch = 1 to epochs do
    Train CNNResBiGRU component..
    for child_epoch = 1 to 5 do
        Train $w_{\mathbb{D}}$ & $w_{\mathcal{C}}$. Fix $w_{\mathcal{R}}$.
        Evaluate performance on validation set
        IF early stopping criteria met THEN terminate training
    end for

    Train MiniRocket component.
    for child_epoch = 1 to 5 do
        Train $w_{\mathcal{R}}$ & $w_{\mathcal{C}}$. Fix $w_{\mathbb{D}}$.
        Evaluate performance on validation set
        IF early stopping criteria met THEN terminate training
    end for
end for

Output: Trained Hybrid Rocket
```

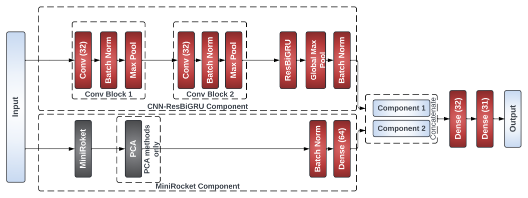


Figure 4.10: Hybrid Rocket architecture combining the existing CNN-ResBiGRU and MiniRocket architectures. Gray coloured layers denote non-trainable transforms. PCA step included in selected hybrid models only.

5 Results

After detailing the initial preprocessing steps for the MyoGym dataset in Chapter 3 and outlining the TSC methods to be applied in Chapter 4, we now offer an analysis of the reported results.

5.1 Reporting the Results

Each method was configured to make a single prediction, as a trainer cannot engage in two activities simultaneously, and windows containing activity change points were excluded. For nearest-neighbor-based methods, making a single prediction involved simply selecting the majority class prediction. In methods utilising neural networks, the final layer was always configured as a fully connected layer with $\mathcal{C}$ classes and a softmax activation. This setup allows the output neurons in each row to be interpreted as a probability distribution.

The paper Wang et al. (2016) uses the Mean Per Class Error (MPCE) to evaluate its methods, calculating the misclassification rate for each class and then averaging them. This metric is particularly useful for addressing class imbalance, as it prevents dominant classes from disproportionately influencing the final score, providing an unbiased assessment of classifier performance, especially for minority classes. However, after downsampling the null activity class, our data is nearly balanced, and we observed no significant difference between MPCE and accuracy. Since the neural networks are optimised using categorical cross-entropy, which closely aligns with accuracy insofar that it does not make any special adjustments for class imbalances, we report accuracy as the baseline metric. We also report a confusion matrix for each method tested in Appendix 7.2.

5.2 Analysing the Results

The table of results provides the test performance of each method. MyoGym includes 31 distinct activities, it's useful to compare these results against a baseline classifier, such as a random guess, which would achieve approximately 3% accuracy with sufficiently large training data. It's clear that all tested methods significantly outperform the random guess, although the discretisation methods have the weakest performance. This may be due to the loss of information inherent in the discretisation process rather than a reflection of the underlying TSC method. While discretisation methods were among the least computationally expensive, shapelet-based methods were the most expensive, yet this did not translate into superior performance. The Learning Shapelets classifier achieved an accuracy of 52.43%, only slightly surpassing the Extracting Shapelets classifier, which achieved 46.91%, despite the latter being trained on a smaller subset of the data due to

[1]The Extracting Shapelets classifier was trained on a subset of $\approx \frac{1}{8}$ of the train data due to the high computational cost associated with this method. As a result, it would likely outperform the test accuracy reported here when trained on the full train data.

Time Series Classifier	**Test Accuracy**
Pairwise DTW Distances	71.55%
DBA	63.65%
SDBA	66.13%
Extracting Shapelets [1]	46.91%
Learning Shapelets	52.43%
SAX + Bag of Patterns	43.67%
SAX VSM	45.86%
Rocket	80.97%
MiniRocket	76.78%
Fully Connected Network	50.14%
InceptionTime (Ensemble)	61.18%
CNN-ResBiGRU	74.41%
Full Hybrid Rocket	74.41%
Full Hybrid PCA	75.26%
Component Hybrid Rocket	75.36%
Component Hybrid PCA	71.74%

Table 5.1: Performance of fitted methods on validation and test datasets

its high computational cost and the former not being limited to finding shapelets present in the data.

The DTW based methods performed well, with DBA and SDBA achieving accuracies of 63.65% and 66.13% respectively. The brute-force pairwise DTW classifier outperformed both, with a 71.55% accuracy, which is expected since each test prediction is made by computing distances to the entire training set. The robust performance of DBA and SDBA is particularly noteworthy, as they rely on selecting a single full-length time series (the barycenter) to represent each activity for predictions on new samples. This is in contrast to the shapelet-based methods, where optimal performance was achieved by using feature vectors derived from $K \approx 50$ of the most informative shapelets with window sizes of 5 to 8, yet even this performance could not beat DBA and SDBA. It is important to note that the high computational cost of the Extracting Shapelets method limited the resolution at which a grid search for shapelet parameters (e.g., number of shapelets, window sizes) could be conducted, whereas the grid search for Learning Shapelets was guided by the Grabock heuristic from Grabocka et al. (2014).

The Rocket and MiniRocket methods achieved accuracies of 80.97% and 76.78% while training at *Rocket speed* compared to other methods. The number of random kernels was set to 10,000, aligning with the MiniRocket paper's (Dempster et al. (2020)) sensitivity analysis, which showed performance improvements plateauing at this feature count across 10 fold validation runs on 40 development datasets - a result replicated in our experiments. The non-trainable nature of the transforms allowed us to fix them during experiments and conduct a grid search for the optimal penalty coefficient α in the log space $[-3, 3]$. MiniRocket reuses different permutations of the same kernel, ensuring that PPV features are on a consistent scale, leading us to find an optimal penalty of $\alpha \approx 2.15$. On the contrary, while Rocket kernels are similarly scaled, the features they generate are not. Specifically, the Global Max Pooling features are on a much larger scale than the PPV features, leading to an initial penalty of $\alpha = 1000$, effectively nullifying the PPV weights and potentially throttling the model's own performance. We deviated slightly from Dempster et al. (2019) by applying standard scaling to each feature before applying Ridge Regression, ultimately selecting $\alpha = 46.42$, which yielded the reported results.

Epochs represent the number of times a neural network pass through the training data to learn the target classifier. The optimal number of epochs varies depending on the context, but in this work, all networks were trained for up to 50 epochs across all experiments. We implemented early stopping, terminating training when validation accuracy stopped improving for a specified number of iterations. The default learning rate was set to 10^{-3} in most experiments, with a callback mechanism to reduce the learning rate when the training loss plateaued, allowing the model to fine-tune the weights more precisely as it approaches the minima.

We applied this approach to Fully Connected Networks, achieving a test accuracy of 50.14%, and InceptionTime (Ensemble), which achieved 61.18%. Notably, test accuracy increased to 74.41% with the CNN-ResBiGRU architecture, the first neural network in our experiments to directly embed a recurrent layer with gating mechanisms for capturing long-term dependencies. In fact, amongst all methods evaluated, there is a premium on those designed to incorporate long term dependencies. The worst performing methods are exclusively those focussed only on short term dependencies: discretisation based methods that create words from short sliding windows, shapelets that extract brief subsequences, and two convolutional architectures lacking pooling or dilation, which would have expanded the receptive field of the kernels. This underscores the importance of modeling long-term dependencies in constructing an effective classifier.

We cannot definitively conclude whether joint or component-wise optimisation is inherently superior as the performances of each proposed hybrid method closely matched. The best-performing hybrid model was the Component Hybrid Rocket, with a test accuracy of 75.36%. No hybrid method showed a significant performance difference compared to the individual component models. Since the outputs of both components are on the same scale, one component's influence should not overshadow the other in the final model predictions. To confirm this, we plotted the learned weights in the final fully connected layer, which combines the 64 output channels from each component, as shown in Figure 5.1.

In Appendix 7.2, we present the confusion matrices for each method where the predicted activity is along the x-axis. The key difference between the best and worst performing methods lies in how often they made clearly erroneous predictions for activities that were not biomechanically similar to the true activity. This likely stems from a lack of memory for long-term dependencies; the more time steps a method can analyse, the more accurately it can distinguish between different time series. Discretisation-based methods exhibited the most frequent misclassifications, while shapelet and neural network methods performed somewhat better. As the highest-performing method, Rocket's confusion matrix merits further validation against our knowledge of biomechanical similarity. This would help confirm that the bottleneck to improved performance is the high similarity between signals from different activities, rather than a specific training issue. For instance, we observed 30 instances where Dumbbell Flyes were confused with Incline Dumbbell Flyes, which are biomechanically identical except for the incline. Additionally, counting the off-diagonal entries in the block for bicep exercises reveals 43 cases of confused activities, compared to just 8 cases where bicep exercises were confused with

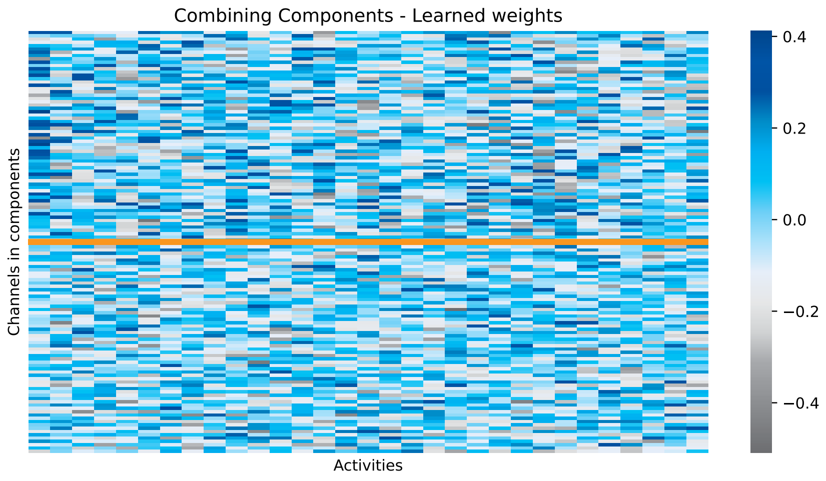


Figure 5.1: Learned weights combining the input channels in the Full Hybrid PCA model. Weights from the 64 PCA component channels sit above the orange line and weights from the 64 CNN-ResBiGRU component channels sit below it.

those targeting different muscle groups. This should reassure the reader that some activities are inherently similar, making their distinction a non-trivial task. The proximity of MiniRocket to what seems to be a performance ceiling, as seen in the confusion matrix, likely accounts for Hybrid Rocket's inability to markedly improve performance.

5.3 Limitations of Results

In Chapter 3, we outlined the separation of the validation and test sets. The validation set is used to fine-tune hyperparameters to produce the best model, but these optimal parameters are of course feature-dependent. For most methods, this isn't an issue since the features are derived directly from the data. However, Rocket-based methods (including MiniRocket, which isn't entirely deterministic) and Hybrid Rocket methods present a double optimisation problem because the features themselves are generated from randomly initialized kernels. Although all Rocket/Hybrid Rocket methods in these experiments used the same set of MiniRocket features, using different sets would require repeating the fine-tuning process and updating predictions accordingly. This would introduce performance variability, which ideally should be measured with confidence intervals across multiple model fits using different random kernel initialisations. Since this work relies on a single set of MiniRocket features, it is limited to providing point estimates only.

Given that methods best incorporating long-term dependencies outperformed those that did not, the window size can be considered a hyperparameter for these approaches. However, in this work, it is fixed at a length of 150, as explained in Chapter 3. There is evidence that pre-setting the window size may have limited these models' ability to learn from the additional information that longer windows could have provided. Nevertheless, allowing the window size to vary across methods would have made direct comparisons between them more difficult.

6 Conclusions and Future Research

This work began with a review of the time series classification literature, establishing the foundational concepts in Chapter 2. Given the challenges in accurately collecting GAR data, in Chapter 3, we proposed applying the MyoGym dataset in our work, a publicly available multivariate dataset capturing time series data from 10 trainers performing various gym activities. We preprocessed this dataset by segmenting it into short windows approximating the length of a single repetition, scaling the channels, and downsampling the dominant background activity class. Initially, we categorized methods in the literature as either classical or neural network-based, but in Chapter 4, we identified a distinct group of time series classifiers that transform data into a time-independent format. In Chapter 5, we applied all the methods introduced in Chapter 4 to the preprocessed MyoGym dataset, evaluating the effectiveness of these transformations. We then proposed embedding them as components of hybrid multi-modal neural networks to determine whether integrate existing architectures into a unified hybrid model could improve their classification performance. Our results demonstrate that these hybrid models perform on par with existing architectures, learning features from both components without either dominating. We conclude that the two components primarily provide the same information, leading to final predictions in the hybrid network that are equivalent to those of either standalone component.

The computational efficiency of the hybrid models would allow us to adopt sliding window steps of 0.5 seconds, as introduced in Chapter 3, technically enabling new predictions every 0.5 seconds in a deployment environment on a smart device with the necessary sensors. While a test accuracy of 75% is significantly better than random guessing, if a trainer reviews their workout time based on predicted activity windows, the 25% misclassification rate could lead to confusing results. In practice, we would smooth consecutive predictions, accounting for the fact that users can't switch activities every 0.5 seconds.

Before deployment, the hybrid model requires refinements. First, rather than downsampling the null activity class as in 3, we could reframe the problem as a two-step process: determine if the trainer is performing an activity, and if so, identify the specific activity. This approach would include all samples and achieve class balance. Additionally, to address the accuracy bottleneck discussed in Chapter 5, predictions could be made at the muscle group level, given that most misclassifications occurred within the same muscle group.

The double optimisation problem over the neural network and the selection of kernel lengths, weights, biases and dilations, as discussed in Chapter 5, presents an intriguing research direction. A search methodology could be employed to sort through the kernel space. This would involve generating features using the kernels, applying Ridge Classification, and then discarding kernels with heavily penalised weights. Alternatively, gradient-based learning methods could be applied to optimise the kernels. Drawing inspiration from the *Learning Shapelets* method in Chapter 4, the research would need to define Rocket features which are smooth and differentiable; for instance, the maximum operation in the Global Max Pooling feature could be redefined using a Soft-max function.

In summary, we have investigated the integration of state-of-the-art time series classifiers from the literature into a unified hybrid multi-modal neural network, discovering that both approaches, despite their distinct architectures, can effectively contribute to the final hybrid model. Although time series classification techniques are typically computationally intensive, we believe that Rocket-based methods hold much promise for reducing the computational demands of time series classification while still delivering top-tier performance.

7 Appendices

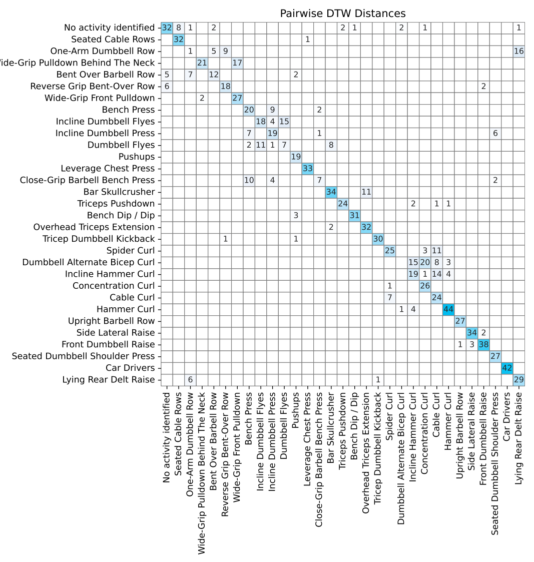
7.1 Appendix 1: MyoGym activities performed

For each of the 30 free weight based gym exercises, we list here: the name of the exercise, the muscle groups the exercises use, the posture the trainer should be in to complete the exercise and the equipment needed.

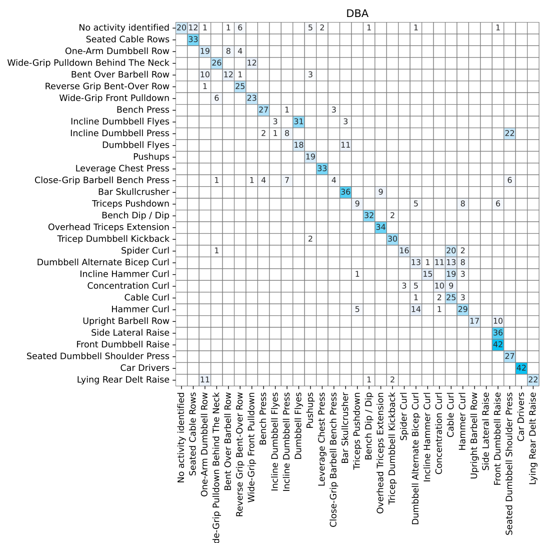
Class Name	**Muscle Group**	**Posture**	**Equipment**
Seated Cable Rows	Middle Back	Seated	Cable
One-Arm Dumbbell Row	Middle Back	Bent Over	Dumbbell
Wide-Grip Pulldown Behind The Neck	Lats	Sitting	Cable
Bent Over Barbell Row	Middle Back	Bent Over	Barbell
Reverse Grip Bent-Over Row	Middle Back	Bent Over	Barbell
Wide-Grip Front Pulldown	Lats	Sitting	Cable
Bench Press	Chest	On back	Barbell
Incline Dumbbell Flyes	Chest	Seated inclined	Dumbbell
Incline Dumbbell Press	Chest	Seated inclined	Dumbbell
Dumbbell Flyes	Chest	On back	Dumbbell
Pushups	Chest	On hands & knees	Own weight
Leverage Chest Press	Chest	Seated	Machine
Close-Grip Barbell Bench Press	Triceps	On back	Barbell
Bar Skullcrusher	Triceps	On back	Barbell
Triceps Pushdown	Triceps	Standing	Cable rope
Bench Dip / Dip	Triceps	Weight on hands	Own weight
Overhead Triceps Extension	Triceps	Standing	Barbell Plate
Tricep Dumbbell Kickback	Triceps	Bent Over	Dumbbell
Spider Curl	Biceps	Sitting	E-Z Curl Bar
Dumbbell Alternate Bicep Curl	Biceps	Standing	Dumbbell
Incline Hammer Curl	Biceps	Seated inclined	Dumbbell
Concentration Curl	Biceps	Seated	Dumbbell
Cable Curl	Biceps	Standing	Cable Bar
Hammer Curl	Biceps	Standing	Dumbbell
Upright Barbell Row	Shoulders	Standing	Barbell
Side Lateral Raise	Shoulders	Standing	Dumbbell
Front Dumbbell Raise	Shoulders	Standing	Dumbbell
Seated Dumbbell Shoulder Press	Shoulders	Seated	Dumbbell
Car Drivers	Shoulders	Standing	Barbell Plate
Lying Rear Delt Raise	Shoulders	On stomach	Dumbbell

7.2 Appendix 2: Confusion Matrices

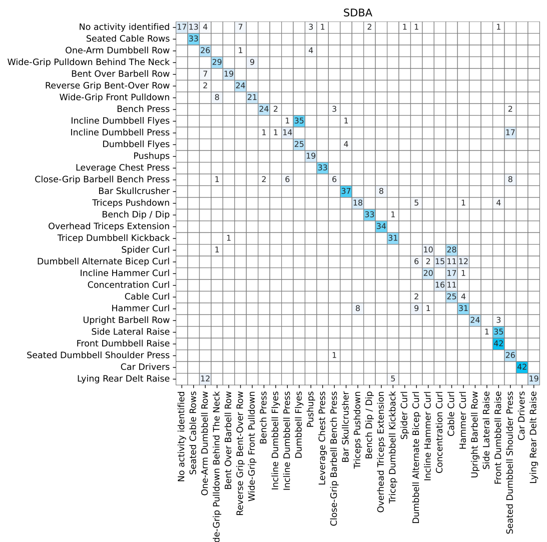
For every method tested in chapter 4, we plot a confusion matrix for the 31 activities in MyoGym. The matrices were processed uniformly, maintaining a consistent color scheme and scaling, with a mask applied to colour only the cells with nonzero counts.



(a) Pairwise DTW Distances

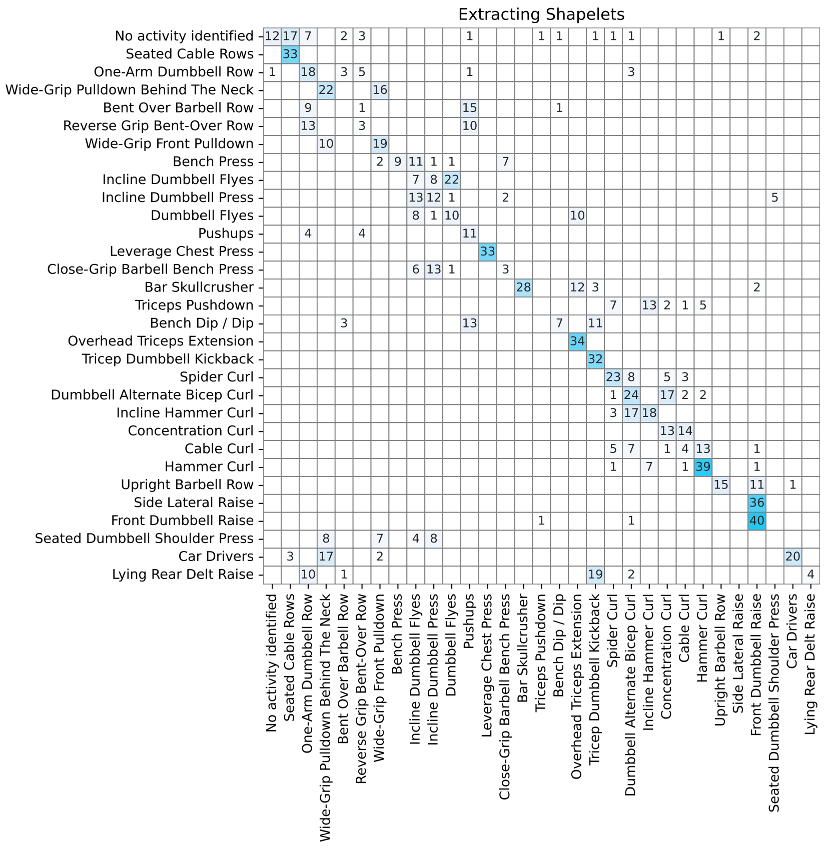


(b) DBA

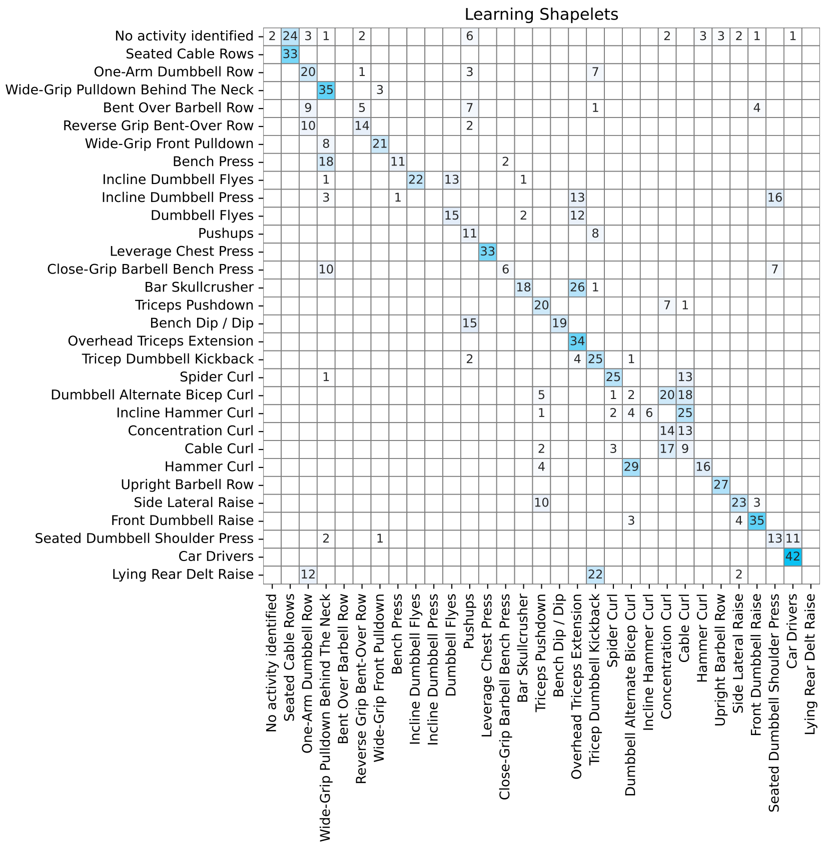


(c) SDBA

Figure 7.1: Confusion matrices for DTW based methods

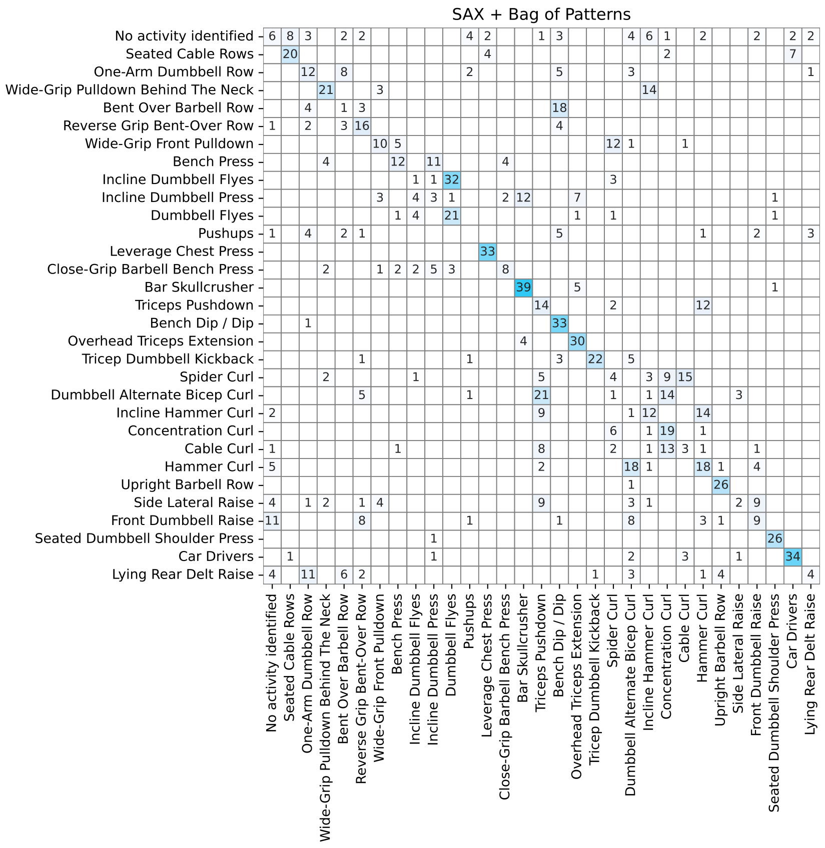


(a) Extracting Shapelets

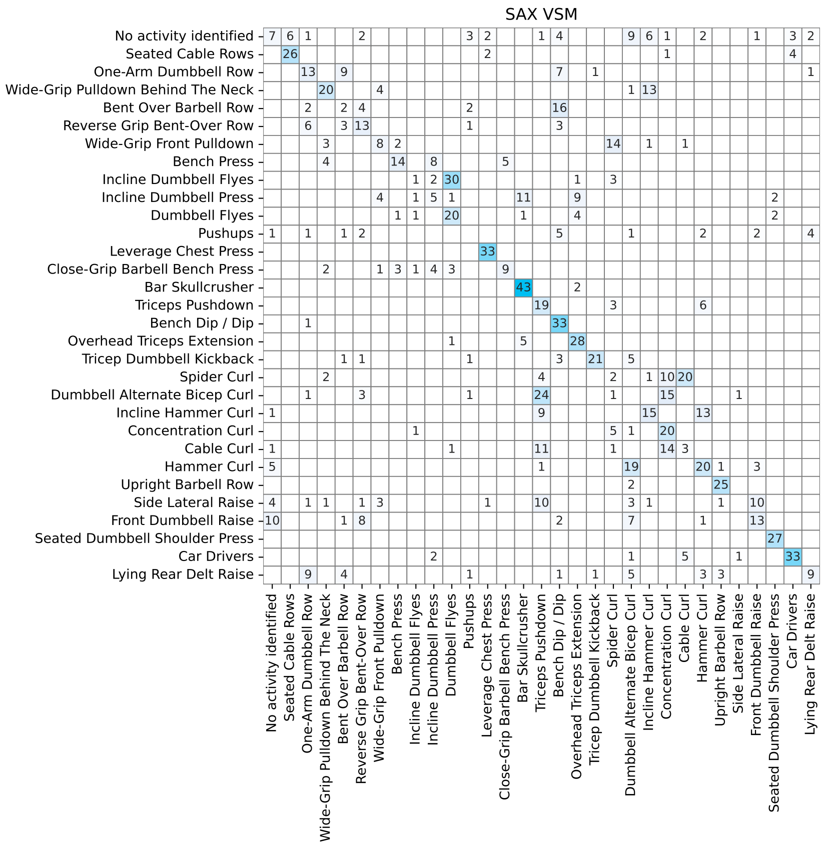


(b) Learning Shapelets

Figure 7.2: Confusion matrices for shapelet based methods

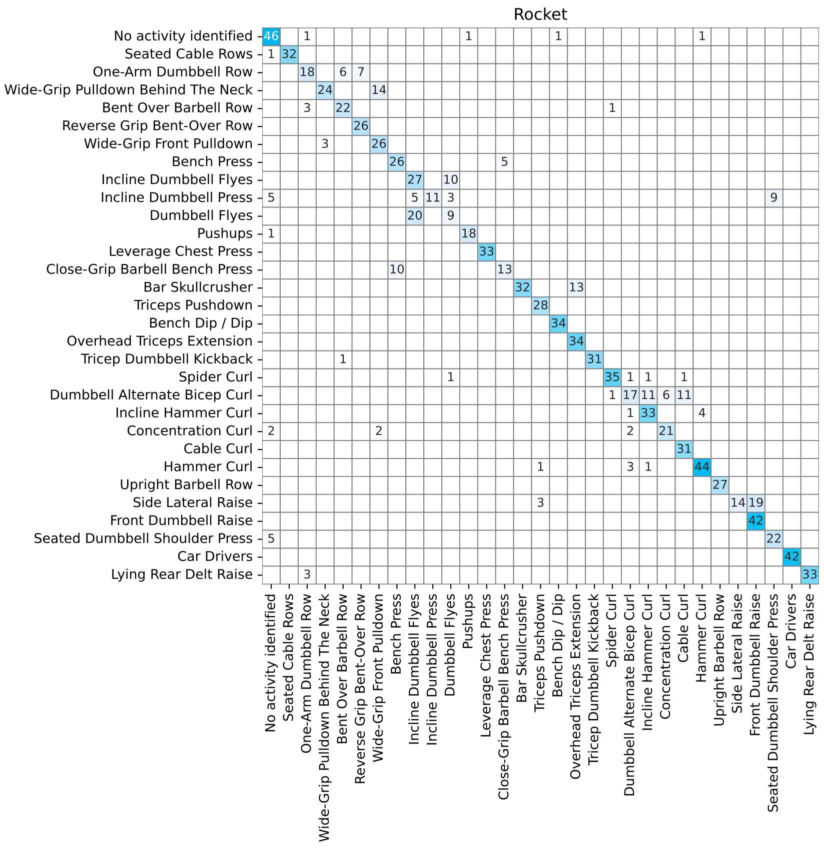


(a) SAX + Bag of Patterns

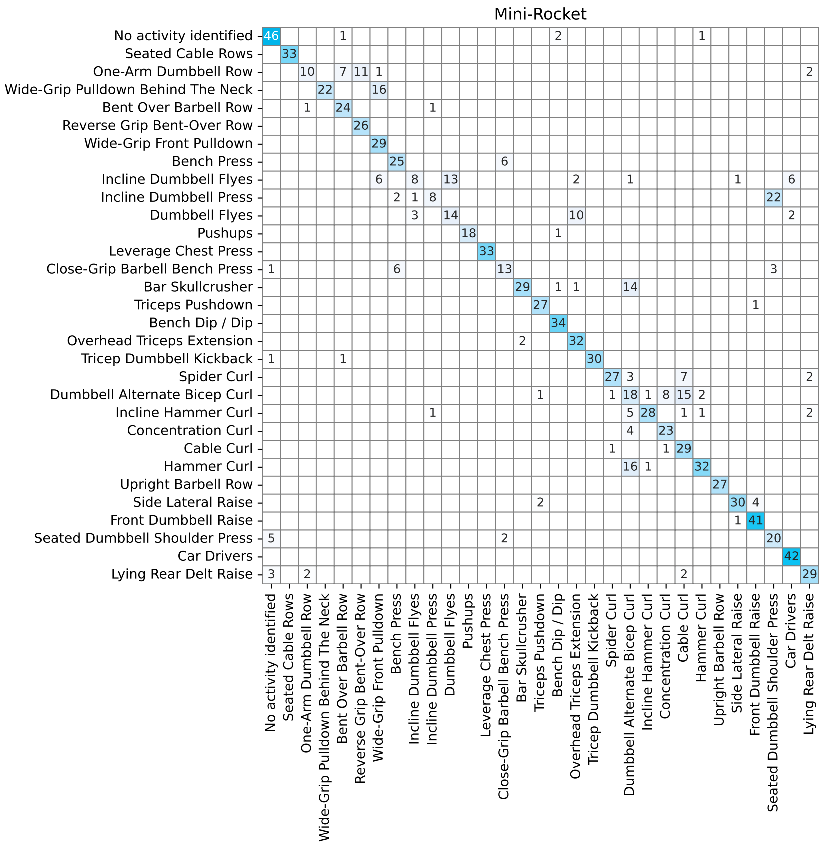


(b) SAX VSM

Figure 7.3: Confusion matrices for discretisation methods

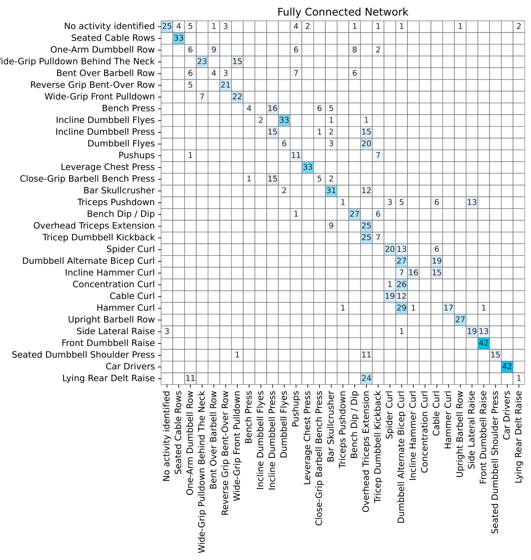


(a) Rocket

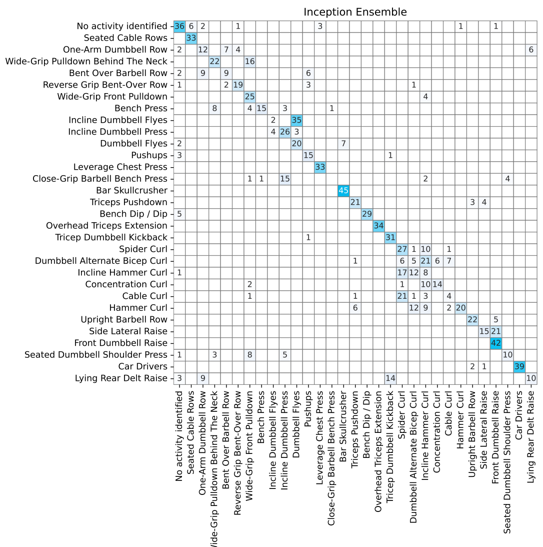


(b) MiniRocket

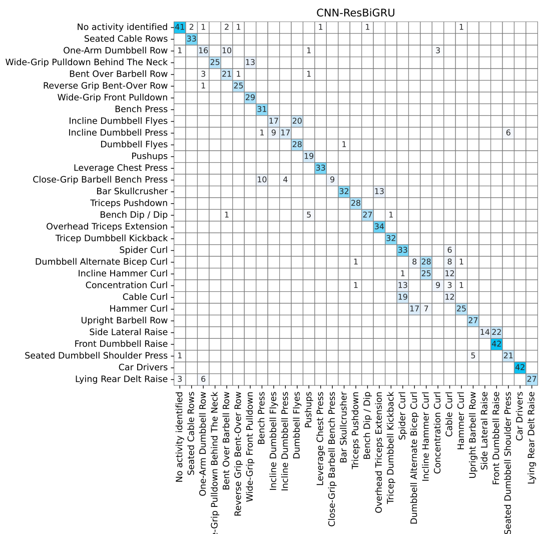
Figure 7.4: Confusion matrices for Rocket methods



(a) Fully Connected Network

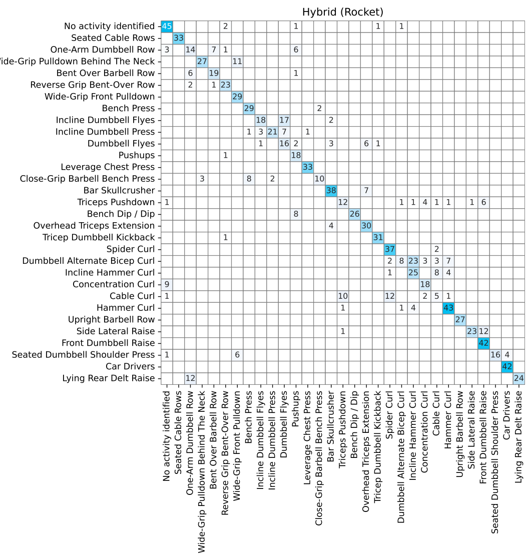


(b) Inception

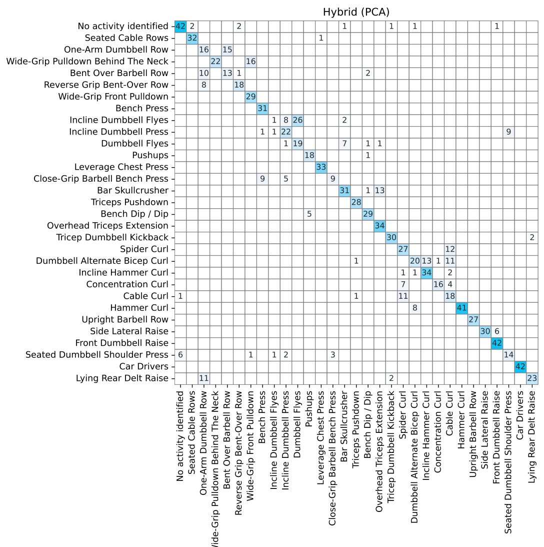


(c) CNNResBiGRU

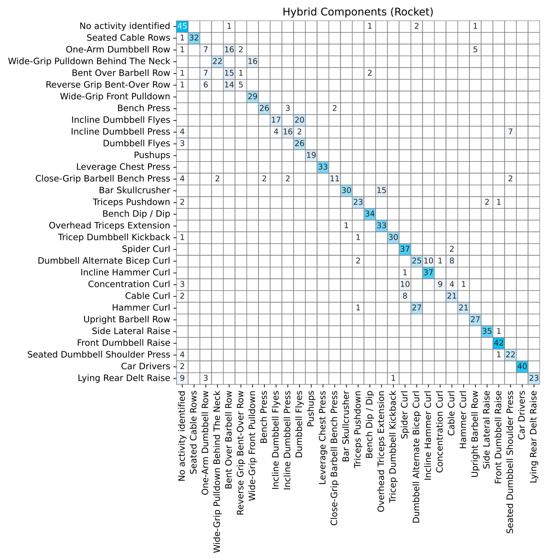
Figure 7.5: Confusion matrices for Neural Network based methods



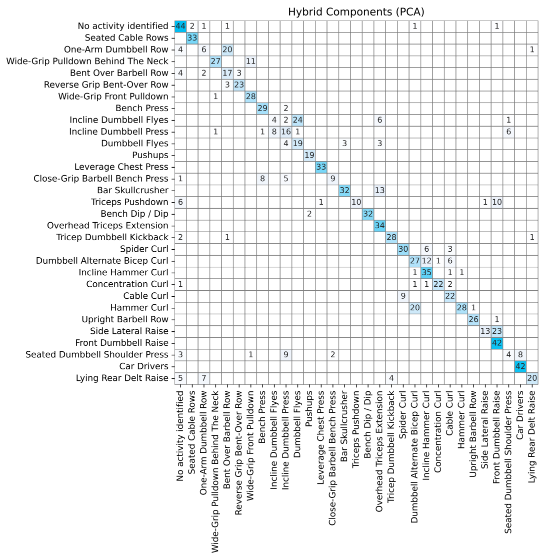
(a) Hybrid (Rocket)



(b) Hybrid (PCA)



(c) Hybrid Components (Rocket)



(d) Hybrid Components (PCA)

Figure 7.6: Confusion matrices for Hybrid Rocket methods

Bibliography

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing*, 2014. URL `https://api.semanticscholar.org/CorpusID:5590763`.

Marco Cuturi and Mathieu Blondel. Soft-dtw: a differentiable loss function for time-series. In *International Conference on Machine Learning*, 2017. URL `https://api.semanticscholar.org/CorpusID:9566599`.

Angus Dempster, Franccois Petitjean, and Geoffrey I. Webb. Rocket: exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery*, 34:1454 – 1495, 2019. URL `https://api.semanticscholar.org/CorpusID:204949593`.

Angus Dempster, Daniel F. Schmidt, and Geoffrey I. Webb. Minirocket: A very fast (almost) deterministic transform for time series classification. *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2020. URL `https://api.semanticscholar.org/CorpusID:229221448`.

Johann Faouzi. Time Series Classification: A review of Algorithms and Implementations. Proud Pen, 2022. URL `https://inria.hal.science/hal-03558165`.

Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, 33:917 – 963, 2018. URL `https://api.semanticscholar.org/CorpusID:52195012`.

Hassan Ismail Fawaz, Benjamin Lucas, Germain Forestier, Charlotte Pelletier, Daniel F. Schmidt, Jonathan Weber, Geoffrey I. Webb, Lhassane Idoumghar, Pierre-Alain Muller, and Franccois Petitjean. Inceptiontime: Finding alexnet for time series classification. *Data Mining and Knowledge Discovery*, 34:1936 – 1962, 2019. URL `https://api.semanticscholar.org/CorpusID:202572652`.

Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *European Conference on Computational Learning Theory*, 1997. URL `https://api.semanticscholar.org/CorpusID:6644398`.

Josif Grabocka, Nicolas Schilling, Martin Wistuba, and Lars Schmidt-Thieme. Learning time-series shapelets. *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014. URL `https://api.semanticscholar.org/CorpusID:207214410`.

Keng hao Chang, Mike Y. Chen, and John F. Canny. Tracking free-weight exercises. In *Ubiquitous Computing*, 2007. URL `https://api.semanticscholar.org/CorpusID:14409288`.

Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2015. URL `https://api.semanticscholar.org/CorpusID:206594692`.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9: 1735–1780, 1997. URL `https://api.semanticscholar.org/CorpusID:1915014`.

Fumitada Itakura. Minimum prediction residual principle applied to speech recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 23:154–158, 1975. URL `https://api.semanticscholar.org/CorpusID:61601418`.

Jason, Luke M. Davis, Jon Hills, and A. Bagnall. A shapelet transform for time series classification. 2015. URL `https://api.semanticscholar.org/CorpusID:9639040`.

Jingwen Jiang, Bryan T. Kelly, and Dacheng Xiu. (re-)imag(in)ing price trends. *Capital Markets: Asset Pricing & Valuation eJournal*, 2020. URL `https://api.semanticscholar.org/CorpusID:234571893`.

Eamonn J. Keogh and Thanawin Rakthanmanon. Fast shapelets: A scalable algorithm for discovering time series shapelets. In *SDM*, 2013. URL `https://api.semanticscholar.org/CorpusID:11759273`.

Heli Koskimäki, Pekka Siirtola, and Juha Röning. Myogym: introducing an open gym data set for activity recognition collected using myo armband. *Proceedings of the 2017 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2017 ACM International Symposium on Wearable Computers*, 2017. URL `https://api.semanticscholar.org/CorpusID:33661857`.

Jessica Lin, Eamonn J. Keogh, Li Wei, and Stefano Lonardi. Experiencing sax: a novel symbolic representation of time series. *Data Mining and Knowledge Discovery*, 15:107–144, 2007. URL `https://api.semanticscholar.org/CorpusID:979006`.

Jessica Lin, Rohan Khade, and Yuan Li. Rotation-invariant similarity in time series using bag-of-patterns representation. *Journal of Intelligent Information Systems*, 39:287 – 315, 2012. URL `https://api.semanticscholar.org/CorpusID:873260`.

Jason Lines, Luke M. Davis, Jon Hills, and A. Bagnall. A shapelet transform for time series classification. In *Knowledge Discovery and Data Mining*, 2012. URL `https://api.semanticscholar.org/CorpusID:2505904`.

Sakorn Mekruksavanich and Anuchit Jitpattanakul. A residual deep learning method for accurate and efficient recognition of gym exercise activities using electromyography and imu sensors. *Applied System Innovation*, 2024. URL `https://api.semanticscholar.org/CorpusID:270936793`.

Matthew Middlehurst, James Large, Michael Flynn, Jason Lines, Aaron George Bostrom, and A. Bagnall. Hive-cote 2.0: a new meta ensemble for time series classification. *Machine Learning*, 110:3211 – 3243, 2021. URL `https://api.semanticscholar.org/CorpusID:233240754`.

Igor Pernek, Karin Anna Hummel, and Peter Kokol. Exercise repetition detection for resistance training based on smartphones. *Personal and Ubiquitous Computing*, 17:771–782, 2013. URL `https://api.semanticscholar.org/CorpusID:20635`.

François Petitjean, Alain Ketterlin, and Pierre Gançarski. A global averaging method for dynamic time warping, with applications to clustering. *Pattern Recognit.*, 44:678–693, 2011. URL `https://api.semanticscholar.org/CorpusID:14850691`.

Martin Rau. Multidimensional binary search trees. *Arch. Formal Proofs*, 2019, 2019. URL `https://api.semanticscholar.org/CorpusID:195769656`.

Aiden Rushbrooke, Jordan Tsigarides, Saber Sami, and Anthony J. Bagnall. Time series classification of electroencephalography data. In *International Work-Conference on Artificial and Natural Neural Networks*, 2023. URL `https://api.semanticscholar.org/CorpusID:263706265`.

Hiroaki Sakoe. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26:159–165, 1978. URL `https://api.semanticscholar.org/CorpusID:17900407`.

Patrick Schäfer. The boss is concerned with time series classification in the presence of noise. *Data Mining and Knowledge Discovery*, 29:1505–1530, 2015. URL `https://api.semanticscholar.org/CorpusID:8915840`.

Patrick Schäfer. Scalable time series classification. *Data Mining and Knowledge Discovery*, 30: 1273–1298, 2016. URL `https://api.semanticscholar.org/CorpusID:9841477`.

Patrick Schäfer and Mikael Högqvist. Sfa: a symbolic fourier approximation and index for similarity search in high dimensional datasets. In *International Conference on Extending Database Technology*, 2012. URL `https://api.semanticscholar.org/CorpusID:14579938`.

Pavel Senin and Sergey Malinchik. Sax-vsm: Interpretable time series classification using sax and vector space model. *2013 IEEE 13th International Conference on Data Mining*, pages 1175–1180, 2013. URL `https://api.semanticscholar.org/CorpusID:15401368`.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014. URL `https://arxiv.org/abs/1409.4842`.

Chang Wei Tan, Angus Dempster, C. Bergmeir, and Geoffrey I. Webb. Multirocket: multiple pooling operators and transformations for fast and effective time series classification. *Data Mining and Knowledge Discovery*, 36:1623 – 1646, 2021. URL `https://api.semanticscholar.org/CorpusID:238198754`.

Zhiguang Wang, Weizhong Yan, and Tim Oates. Time series classification from scratch with deep neural networks: A strong baseline. *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 1578–1585, 2016. URL `https://api.semanticscholar.org/CorpusID:14303613`.

Jianbo Yang, Minh Nhut Nguyen, Phyo Phyo San, Xiaoli Li, and Shonali Krishnaswamy. Deep convolutional neural networks on multichannel time series for human activity recognition. In *International Joint Conference on Artificial Intelligence*, 2015. URL `https://api.semanticscholar.org/CorpusID:1605434`.