Scalable Classification of Univariate and Multivariate Time Series

Saeed Karimi-Bidhendi, *Student Member*, *IEEE*
Electrical Engineering and Computer Science
University of California, Irvine
Irvine, United States
skarimib@uci.edu

Faramarz Munshi
SOAS
United Kingdom
faramarzmunshi@gmail.com

Ashfaq Munshi
PepperData Inc.
United States
ashfaqmunshi@pepperdata.com

***Abstract*—Time Series Classification (TSC) is important in many applications including IoT, medical, stock market analysis, economic forecasting, process and quality control and Big Data systems. The problem of time series classification has been studied separately for univariate (UTS) and multivariate (MTS) time series using different datasets and techniques. In this paper, we propose a unique, off-the-shelf approach to classifying time series that improves the current state-of-the-art accuracy for UTS and its generalization to MTS. Our technique maps each time series to a Gramian Angular Difference Field (GADF), interprets that as an image and uses Google's pre-trained Convolutional Neural Network (trained on Inception v3) to map the GADF images into a 2048-dimensional vector space. Then, a multilayer perceptron (MLP) with three hidden layers, and a softmax activation function at the output is used to achieve the final classification. Our method yields competitive results for training and prediction in the UTS case while delivering superior results for MTS datasets. Unlike many published results, our technique is robust in the presence of variable length time series with missing data points, and scales well with the size of dataset.**

***Keywords*-Time Series, Classification, Neural Networks, Deep Learning, Big Data.**

I. Introduction

Time Series (TS) occur naturally in many domains such as IoT, medical, stock market analysis, economic forecasting, process and quality control, utility studies and Big Data systems. Some series, such as heart rhythm, are studied as univariate TS while others, such as those extracted from sensors on a robot, are inherently multivariate. Our interest comes from TS data that are gathered from applications running on hundreds and thousands of nodes on Big Data systems at a variety of Fortune 200 companies.

At Pepperdata Inc. (PD), we have a metric system that continuously instruments every Java Virtual Machine (JVM) on every node of a Big Data cluster. This metric system collects over 300 data points, such as CPU, I/O, memory, and garbage collection data, every five seconds from every application running on these clusters. Each of these data points describes a unique TS. As an example, a 570 node cluster at a customer site generates 41 million points per minute. Overall, company PD currently collects over one billion data points per minute across all our customers with this number increasing nearly 25% every quarter. Big Data clusters are typically hundreds, if not thousands, of nodes and represent a large capital expenditure on the order of tens of millions of dollars. An important goal is therefore to optimally utilize the cluster assets and to ensure that all applications are scheduled as efficiently as possible. To do this, we need to understand the behavior of these applications and how they affect critical machine resources so that the scheduler, whether human or Apache Hadoop's components such as the YARN resource manager and HDFS distributed file system, can make optimal choices.

Classifying the TS of the underlying data is critical towards providing hints to the scheduler so that it can act, while knowing the structure of resource utilization that an application will have while it is running. The data points that were used in our experiments were a selection of five from over 300 data points collected every five seconds from every application running on the cluster. Each data point is a unique TS that describes the particular usage of the metric as a percentage of the total cluster or as a number of bytes. The five metrics we used were as follows: the number of active tasks of a given application, the CPU usage as a percentage of the total cluster used by the application, the Virtual Core RAM usage as a number of bytes used by the application, the number of bytes read from the network by the application, and the number of bytes read from HDFS by-the-application. Each application has a different runtime, and each metric may not be recorded or may appear as missing data points if it is under certain threshold. As a result, each TS has a varied length and contains missing values. Nodes could end up not reporting for long periods of time whether because of network issues or hardware failures, creating even more missing and noisy data. A single TS, then, for an application could be as little as 10 to as many as 10,000 data points, containing as few as 0 missing data points to as many as 400. Our goal is to classify this collection of TS to give operators a better understanding of resource utilization on their clusters and to enable a scheduler to better optimize cluster resources. Below, we describe our approach.

II. Univariate Time Series Classification

The subject of univariate TSC has received much attention over the past decade and many methods have been devel-

978-1-5386-5035-6/18/$31.00 ©2018 IEEE

oped to improve the classification accuracy. Most of these works are evaluated over the UCR time series repository [8]. Although our proposed model demonstrates competing results for univariate TSC over UCR datasets against state-of-the-art models, it is originally designed to address the challenges that do not exist in UCR benchmark datasets such as large amount of data, missing data points, variable length TS etc. Moreover, we will see later in section III that a slight modification of the same model can be used when each labeled data is inherently multivariate, such as our datasets in company PD.

Formally, a *time series* $\vec{X} = (x_1, x_2, ..., x_n)$ is an ordered sequence of $n \in \mathbb{N}$ real numbers, usually at equally spaced time intervals. The parameter n is referred to as the *length* of the TS $\vec{X}$. We always assume that $n < \infty$, i.e., the TS has finite length.

A. Related Work

Earlier methods focused on indexing TS under Euclidean distance metric [24]. Later, other similarity measures such as Dynamic Time Warping (DTW) were explored to perform the classification [14], [21]. In the last decade, the K-NN classifier proved to be an efficient approach when used DTW as a distance measure between TS; however, today these approaches are commonly used as baselines since they fail for long or noisy TS.

Instead of incorporating the TS data directly, feature-based approaches extract the distinguishing characteristics of the data for each label to train the classifier. These methods either use shapelets which are TS sub-sequences that can be interpreted as maximal representative of a class [15], or bag-of-patterns (BOP) to decompose each TS into a bag of substructures. Two successful shapelet methods are Shapelet Transform (ST) which incorporates the distance to the shapelets as a new set of features [5], [18], and Learning Shapelets (LS) in which a set of optimal shapelets are synthetically constructed [15]. Shapelets provide interpretable and accurate predictions on some datasets [18], but they suffer from high computational complexity. BOP methods create a set of discrete features from substructures of a TS. One popular BOP algorithm is the Time Series Bag-of-Features (TSBF) technique which creates windows of stochastic position and length from the TS and builds a codebook from a random forest classifier [17]. Another proposed BOP framework is Bag-Of-SFA-Symbols (BOSS) which applies the Symbolic Fourier Approximation (SFA) to obtain the discrete features for each sliding window of fixed length over the TS [9]. This method is extended in [10] to a framework called Bag-Of-SFA-Symbols in Vector Space (BOSS VS) which uses a compact representation of classes instead of TS using a vector space model in order to reduce the classification time. Another approach called Word ExtrAction for time SEries cLassification (WEASEL) divides TS into windows and applies Fourier transform and low-pass filter in each window to reduce both noise and dimensionality of the data. Fourier coefficients are quantized as symbols of an alphabet; thus, windows are mapped into words and features are extracted from the bag-of-words (sentences) and passed to a logistic classifier [4].

Above techniques mostly depend on hand-crafted feature engineering. Automated feature extraction was made possible with recent developments of deep learning. In [2] each UTS is mapped to three different images using the *Gramian Angular Fields* (GAF) and *Markov Transition Fields* (MTF). These images are tiled and used to train a tiled Convolutional Neural Network (CNN) for classification. In [1], three neural network models are trained on raw TS: *Multilayer Perceptrons* (MLP), *Fully Convolutional Networks* (FCN) and *Residual Networks* (ResNet). While the FCN achieved a new state-of-the-art result compare to previous approaches, its structure is so immense that it is computationally intractable to train the FCN model on either images introduced in [2], or any large dataset.

Another type of TS classifiers are Ensemble frameworks in which a collection of different classifiers are used to predict the label for each TS under a criterion such as majority voting. The collective of transformation-based ensembles (COTE) and elastic ensemble (EE PROP) are two popular ensemble frameworks that take advantage of 35 and 11 core classifiers, respectively [13], [12]. While these techniques can combine the advantages of each individual core classifier to obtain a superior performance, they cannot be utilized in practice for large datasets due to high training runtime for optimizing all core classifiers.

Here at company PD, we present a simple and scalable model called *Classifier PD* for TSC that not only outperforms previous works on univariate TSC, but can also be generalized to MTS classification in the presence of varied length TS and missing data.

B. Our Method

First, we pre-process the raw data to extract a new and rich set of features. These features are then used for training and prediction in a deep learning module. The whole pipeline is depicted in Fig. 1.

1) Preprocessing and Feature Extraction: The preprocessing consists of two steps. First, we encode each TS of length n as an $n \times n$ image. Then convolutional neural networks are utilized to extract features from these images.

Step 1. Imaging Time Series:

The framework used for encoding TS as image is called the *Gramian Angular Difference Field* (GADF) that is first introduced in [2]. In this method, a time series $\vec{X} = (x_1, x_2, ..., x_n)$ is first normalized to the interval $[-1, 1]$ by:

$$\tilde{x}_i = \frac{2x_i - \max(\vec{X}) - \min(\vec{X})}{\max(\vec{X}) - \min(\vec{X})} \quad , \quad \forall 1 \leq i \leq n \qquad (1)$$

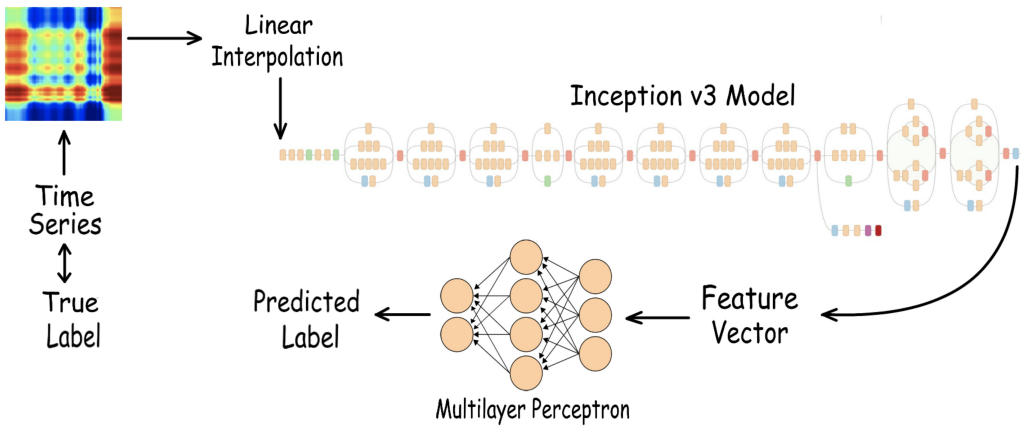


Fig. 1. Classifier PD

While [2] uses the maximum and minimum of each TS to map it into the interval $[-1, 1]$ as in (1), in this work, we use the maximum and minimum of all TS in our training dataset in order to map each TS into $[-1, 1]$ to account for relative magnitude among our labeled data. This strategy makes our model more robust than [2] because the uniformed maximum and minimum makes the model capture the series-wise magnitude which is important for some classes. In another words, we use the following values:

$$\text{MAX} = \max_{\vec{X} \in \text{Training Data}} \vec{X} \quad , \quad \text{MIN} = \min_{\vec{X} \in \text{Training Data}} \vec{X} \quad (2)$$

to map the time series $\vec{X} = (x_1, x_2, ..., x_n)$ to $[-1, 1]$ as:

$$\tilde{x}_i = \frac{2x_i - \text{MAX} - \text{MIN}}{\text{MAX} - \text{MIN}} \quad , \quad \forall 1 \leq i \leq n \qquad (3)$$

Time series in the test dataset are also normalized in the same way. Each data point $\tilde{x}_i$ of the rescaled TS $\tilde{\vec{X}} = (\tilde{x}_1, \tilde{x}_2, ..., \tilde{x}_n)$ is then represented in polar coordinates via the following radius and angle:

$$\phi_i = \arccos(\tilde{x}_i) \qquad , \qquad r_i = \frac{t_i}{n} \qquad (4)$$

where $t_i \in \{1, 2, ..., n\}$ is time stamp. The GADF image is then defined from polar representation of the rescaled TS to be an $n \times n$ matrix as follows:

$$G = \left[\sin\left(\phi_i - \phi_j\right)\right]_{1 \leq i,j \leq n} \qquad (5)$$

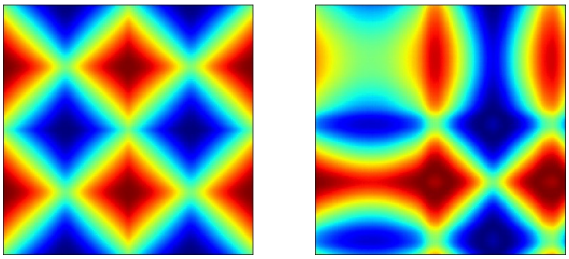
The advantage of GADF image is that it captures both magnitude and time dependencies between data points in a TS. The element $G_{i,j}$ encodes the relative correlation by superposition of directions for data points x_i and x_j at time interval $|i - j|$ when represented in polar coordinates. Furthermore, elements $G_{i,j}$ where $|i - j| = t$ stands for all pairs of data points with time distance t; thus, temporal dependency is preserved as position moves from top-left to bottom-right in the image. As an example, Fig. 2 depicts two GADF images corresponding to two time series with different labels in Automatic Diatom Identification and Classification (ADIAC) time series dataset.

In practice, the original TS may have a few missing data points. If the point x_i in $\vec{X} = (x_1, x_2, ..., x_n)$ is missing, i.e. x_i = NaN (Not a Number), then all elements in the i^{th} row and column of the image G in (5) are NaN as well. Therefore, we need to interpolate these missing pixels of the GADF image. Many sophisticated interpolation techniques can be used for this purpose; however, we utilized a simple linear interpolation on vectorized version of the image G to predict the value of missing pixels. More precisely, we flatten the $n \times n$ image G in (5) into a vector of size $1 \times (n \times n)$ and perform one-dimensional linear interpolation for the missing pixel values. This simple approach turned out to be effective enough in our overall scheme.

Step 2. Image Processing:

Now that each TS is converted to an image, convolutional neural networks are utilized to extract features from these images. CNNs are a family of Artificial Neural Networks (ANNs) that are commonly used in situations where data can be presented as an image or map wherein the proximity between two data points implies how associated they are. The architecture of a CNN is comprised of a series of convolutional and pooling layers in which features of the data are learned in different layers of abstraction. In this paper, instead of training a CNN from scratch, we use the pre-trained Inception v3 model, which is trained for a standard image classification task in computer vision, i.e., the ImageNet Large Visual Recognition Challenge. Not only this approach lets us extract useful features of the images, but also we bypass the overhead for training and optimizing a model for this purpose which in turn significantly improves the training runtime of our method.

In order to use the Inception v3, we remove the top layer of this model that does the final classification; thus, when an image is fed to the Inception v3, the values prior to



(a) (b)

Fig. 2. Sample GADF images for two time series with different labels in ADIAC dataset.

the final classification layer are returned as a new set of features that contain useful information from the image. In order to match the image shape to the input layer size, the Inception v3 model bilinearly resizes the image to 299×299; hence, regardless of the input image size, a vector of 2048 features is always returned from the Inception v3 model. This property implies another advantage of using transfer learning for feature extraction: variable length time series are all mapped into a 2048 dimensional Euclidean space, i.e., all TS have a new fixed length representation. As discussed in the next section, these new features are then used as input to train another neural network to categorize the data.

2) Classification Model: While we explored how to obtain distinguishing characteristics of the TS, we are still to find an input-output mapping that predicts a class index given an input data. To this end, a family of ANNs called Deep Neural Networks (DNNs) are employed in which a series of nonlinear transformations are applied to the input data in order to map it to its corresponding label, where each transformation is referred to as a layer.

The utilization of a DNN as a classifier is comprised of two major steps: First, we need to determine the hyperparameters of the network that define its structure such as the number of nodes and hidden layers, activation functions in each hidden layer, regularization techniques used to avoid overfitting etc. Here, we consider a DNN that consists of three hidden layers. Since output of the Inception v3 serves as input to train the DNN, we have 2048 nodes in its input layer. There are 800, 400 and 100 nodes in the first, second and third hidden layers respectively. The number of nodes in the output layer of DNN is equal to the number of classes in the classification task. In all hidden layers, ReLU function is used as activation. In order to lessen the issue of covariate shift and increase the stability of a neural network during training, batch normalization is applied to the output of each hidden layer by subtracting the batch mean and dividing by the batch standard deviation. Moreover, a dropout rate of 50% is used in all hidden layers that randomly sets neurons on and off so that the DNN model generalizes well and does not overfit on the training data. Finally, a softmax activation function is used in the final layer of DNN to interpret the output values as a set of probabilities over all possible classes. Second, we need to train the DNN to find the proper values for parameters of the network (i.e., weights and biases). For this purpose, we use the Adadelta optimizer in order to minimize the categorical cross entropy loss function between predicted and true labels.

C. Experiments & Results

Here, we evaluate our classifier PD on various TS benchmark datasets of UCR repository, and compare our framework to the best existing TSC methods such as 1-NN DTW and 1-NN DTW CV [12], TSBF [17], COTE [13], LS [15], EE PROP [12], ST [5], BOSS [9] and BOSS VS [10], WEASEL [4], MLP and FCN and ResNet [1]. In Table I, we summarize the performance of classifier PD along with those numbers used by authors in [4] and [1] for test accuracies of 13 other core TSC models on these UCR datasets. Bold numbers indicate the best accuracy rate for each dataset. As it is shown in Table I, our method outperforms previous works on univariate TSC and achieves a new state-of-the-art result on UCR benchmark datasets. Moreover, our classifier achieves higher average accuracy compare to previous state-of-the-art models.

Several high accuracy classifiers, such as BOSS and ST, scale quadratically (power of 2) with the size of training data. Ensemble techniques such as COTE and EE PROP also suffer from high computational cost since they employ many core classifiers and each classifier has to be independently trained. Therefore, it is infeasible to use these models in practice when the dataset is large. Other methods such as 1-NN DTW, BOSS VS and deep learning models (i.e., MLP, FCN and ResNet) scale linearly with the number of training data; hence, these methods are more suitable for large datasets. Note that our method performs the same amount of computation for each time series and thus scales linearly with the number of training data while having a higher accuracy than prior work.

III. Multivariate Time Series Classification

In a broad range of fields such as multimedia, engineering etc. a process is characterized by observations over not one but several variables. These data inherently consist of a finite sequence of UTS, which we refer to as multivariate TS.

Formally, for k univariate time series $\vec{X}_1, \vec{X}_2, ..., \vec{X}_k$ of lengths $n_1, n_2, ..., n_k$, a Multivariate Time Series is an ordered k-tuple $X = (\vec{X}_1, \vec{X}_2, ..., \vec{X}_k)$. The parameter k is referred to as the *dimension* of MTS, and the univariate time series $\vec{X}_i$ is called the i^{th} *component* of X. In this work, we assume that all MTS in both training and test datasets have the same dimensionality, i.e., the parameter k is fixed. However, lengths of the TS within a MTS are completely arbitrary and can be different from those of another MTS as well, i.e., the parameters $n_1, n_2, ..., n_k$ are not necessarily consistent across different MTS.

Table I: Best accuracy rate for various univariate time series classification methods (%)

	1-NN DTW	1-NN DTW CV	TSBF	COTE (ensemble)	Learning Shapelet	EE (PROP)	ST	BOSS	BOSS VS	WEASEL	MLP	FCN	ResNet	PD
Adiac	60.4	61.1	76.98	79.03	56.32	66.5	78.26	76.47	69.8	83.12	75.2	**85.7**	82.6	83.12
ArrowHead	66.3	80	75.43	81.14	84.57	81.14	73.71	83.43	82.9	85.71	82.3	**88**	81.7	84
CBF	99.7	99.4	98.78	99.56	99.44	99.78	97.45	99.78	99.9	98.34	86	**100**	99.4	93.89
Coffee	**100**	**100**	**100**	**100**	**100**	**100**	96.43	**100**	96.4	**100**	**100**	**100**	**100**	**100**
Computers	70	62	75.6	74	58.4	70.8	73.6	75.6	67.6	65.6	54	**84.8**	82.4	73.6
DistPhalOutlineAgeGroup	77	62.6	78.26	76.09	77.9	72.83	77.54	77	84.5	76.98	82.7	83.5	79.8	**85.75**
DistPhalOutlineCorrect	71.7	72.5	71.22	74.82	71.94	69.07	76.98	73.9	71.8	77.9	81	81.2	82	**85.5**
DistPhalTW	59	63.3	67.63	69.78	62.59	64.75	66.19	70.5	74.7	67.63	74.7	**79**	74	**79**
Earthquakes	71.9	72.7	74.82	74.82	74.1	74.1	74.1	74.8	80.7	74.82	79.2	80.1	78.6	**82.3**
ECG5000	75	92.5	93.96	94.6	93.22	93.87	94.38	94.13	89	**94.82**	93.51	94.1	93.1	93.51
Herring	53.1	53.1	64.06	62.5	62.5	57.82	67.19	54.69	59.4	65.63	68.7	**70.31**	59.4	**70.31**
LargeKitchenAppliances	79.5	79.5	52.8	84.53	70.13	81.07	85.87	76.53	69.6	68.27	48	**89.6**	89.3	83.47
Lighting2	86.9	86.9	73.77	86.88	82.3	88.53	73.77	83.61	73.8	55.74	72.1	80.3	75.4	**93.44**
Lighting7	72.6	71.2	72.6	80.82	80.27	76.71	72.6	68.49	71.2	71.23	64.4	**86.3**	83.6	83.56
Meat	93.3	93.3	93.33	91.67	73.33	93.33	85	90	83.3	91.67	93.3	96.7	**100**	**100**
MedicalImages	73.7	74.7	70.53	75.79	72.95	74.21	66.97	71.84	52.6	74.08	72.9	**79.2**	77.2	77.24
MidPhalOutlineAgeGroup	50	51.9	**81.44**	80.41	78.7	78.35	79.38	56.5	74.7	60.4	73.5	76.8	76	79.5
MidPhalOutlineCorrect	69.8	76.6	57.79	63.64	57.14	55.85	64.29	77	65	80.76	76	79.5	79.3	**82.5**
MidPhalTW	50.6	50.6	59.74	57.14	50.65	51.3	51.95	52.6	58.6	53.9	60.9	61.2	60.7	**62.41**
OliveOil	83.3	86.7	83.33	90	44	86.67	90	86.67	86.7	**93.33**	40	83.3	86.7	**93.33**
PhalangesOutlinesCorrect	72.8	76.1	82.98	77.04	76.46	77.27	76.34	77.16	68.3	81.7	83	82.6	82.5	**84.97**
ProxPhalOutlineAgeGroup	80.5	78.5	87.28	86.94	84.88	80.76	**88.32**	84.88	75.6	84.88	82.4	84.9	84.9	**88.32**
ProxPhalOutlineCorrect	78.4	79	84.88	85.37	83.42	80.49	84.39	83.42	86.6	89.69	88.7	90	91.8	**92.1**
ProxPhalTW	75.6	75.6	80.98	78.05	77.56	76.59	80.49	81.5	75.2	80.98	79.7	81	80.7	**82.25**
RefrigerationDevices	46.4	44	47.2	54.67	51.47	43.73	58.13	49.87	51.2	53.87	37.1	53.3	52.8	**58.4**
SonyAIBORobot	72.5	69.6	79.53	84.53	89.74	70.38	84.36	63.23	73.5	82.36	72.7	96.8	**98.5**	90.35
StarlightCurves	90.7	90.5	97.71	97.96	96.65	92.61	97.85	97.78	90.4	97.73	95.7	96.7	97.5	**98.06**
Strawberry	94.1	94.6	95.41	95.14	91.08	94.59	96.22	97.6	97.6	97.57	96.7	96.9	95.8	**97.88**
SwedishLeaf	79.2	84.6	91.52	95.52	91.3	91.52	92.8	92.16	85.9	**96.6**	89.3	**96.6**	95.8	96.16
Symbols	95	93.8	94.57	96.38	96.44	95.98	88.24	96.68	**97.1**	96.18	85.3	96.2	87.2	96.48
Trace	**100**	99	98	**100**	**100**	99	**100**	**100**	**100**	**100**	82	**100**	**100**	**100**
Wafer	98	99.5	99.51	99.98	99.62	99.74	**100**	99.48	99.9	**100**	99.6	99.7	99.7	99.69
#Wins	2	1	2	2	2	1	3	2	2	6	1	12	4	**19**
Average Accuracy	76.47	77.36	80.05	82.78	77.78	79.35	81.03	80.23	78.55	81.3	77.21	86.07	84.64	**86.6**

A. Related Work

A large category of proposed models is mostly concerned with finding an efficient measure to compare MTS. One of the earliest such measures was Euclidean distance; however, this basic metric fails when our TS have variable length. An alternative metric proposed to address this issue was DTW distance between MTS. Although DTW measure was first proposed for UTS, some extension of it to MTS can be found in [25], [20]. DTW distance measure performs well when used with a distance-based classifier such as 1-nearest neighbor (1-NN); thus, $1NN_{DTW}$ is widely used as a baseline for multivariate TSC. A variation of this method called DDTW is proposed in [22] where the distance measure is defined to be the DTW distance between derivatives of MTS. Ref. [11] improves upon both DTW and DDTW by combining them into a parametric distance measure called DD_{DTW}. One disadvantage of DTW is that it cannot stand for correlation between different variables in a MTS and to address this issue, [7] proposed Mahalanobis distance based Dynamic Time Warping (MDDTW) as a measure for multivariate TSC. In [7] two models called $1NN_{MDDTW}$ and SVM_{MDDTW} are proposed in which k-nearest neighbors (k-NN) and support vector machine (SVM) classifiers are utilized along with MDDTW measure. These two methods were shown to outperform many existing models. Ref. [23] introduces a traditional SVM that incorporates a kernel based on DTW, called SVM_{DTW}. A new temporal variant of discrete SVM called TDVM is developed in [19] that not only defines the objective function to account for overall similarity among TS with the same label, but also makes the classifier capable of learning from small number of training instances. Ref. [16] proposed a classifier based on symbolic representation for MTS called SMTS to extract information from all attributes of MTS simultaneously, and also scaled well with the number of attributes which makes it computationally efficient. Recently, a novel deep learning method is introduced in [6] which is based on the recurrent neural network (RNN) and adaptive differential evolution (ADE), called C_{ADE}. This method makes full use of the network state space of a given RNN to induce classifiers rather than training the network, and proved to outperform many existing methods on a variety of standard MTS datasets. Ref. [3] proposed two deep learning models called Multivariate LSTM-FCN (MLSTM-FCN) and Multivariate ALSTM-FCN (MALSTM-

FCN) in which Long Short-Term Memory (LSTM) and Fully Convolutional Network (FCN) are incorporated for both feature extraction and classification. These models are successfully applied on several datasets and improved the results in some cases. In what follows, we describe how a slight modification of our model PD in Fig. 1 can generalize it from UTS to MTS classification task, while preserving its competitive performance.

B. Our Method

Our proposed pipeline for multivariate TSC is very similar to that of univariate TSC described earlier in section II.B. in the sense that we preprocess our dataset by converting each MTS to an image and extract its features via the pre-trained Inception v3. Finally, we use a deep learning model for training and prediction on the preprocessed data.

The same procedure discussed in section II.B. for imaging a UTS can be applied to transform each component of a MTS into a GADF image. Hence, k GADF images are produced for a MTS of dimension k. These images are then concatenated vertically to form one single $k-$channel image. Note that each component of the MTS is converted to an image independently of all other $k-1$ components, and the same procedure for imaging a UTS needs to be repeated for each component separately. More precisely, for any $1 \leq i \leq k$, in order to transform the i^{th} component of the MTS $X = (\vec{X}_1, \vec{X}_2, ..., \vec{X}_k)$, i.e., $\vec{X}_i = (x_{i,1}, x_{i,2}, ..., x_{i,n_i})$ into an image, first we need to normalize it into the interval $[-1, 1]$ as follows:

$$\tilde{x}_{i,j} = \frac{2x_{i,j} - \text{MAX}_i - \text{MIN}_i}{\text{MAX}_i - \text{MIN}_i} \quad , \quad 1 \leq j \leq n_i \tag{6}$$

where MAX_i and MIN_i are the maximum and minimum over the i^{th} component of all MTS in the training dataset:

$$\begin{aligned} \text{MAX}_i &= \max_{i^{th} \text{ component of } X \in \text{Training Data}} \vec{X}_i \\ \text{MIN}_i &= \min_{i^{th} \text{ component of } X \in \text{Training Data}} \vec{X}_i \end{aligned} \tag{7}$$

for all $1 \leq i \leq k$. Multivariate time series in the test dataset are also normalized in a similar way. Each data point $\tilde{x}_{i,j}$ of the rescaled TS $\tilde{\vec{X}}_i = (\tilde{x}_{i,1}, \tilde{x}_{i,2}, ..., \tilde{x}_{i,n_i})$ is then represented in polar coordinates as:

$$\phi_{i,j} = \arccos(\tilde{x}_{i,j}) \quad , \quad r_{i,j} = \frac{t_j}{n_i} \tag{8}$$

where $t_j \in \{1, 2, ..., n_i\}$ is the time stamp. The GADF image for the i^{th} component of X is then defined to be an $n_i \times n_i$ matrix as follows:

$$G_i = [\sin(\phi_{i,j} - \phi_{i,l})]_{1 \leq j,l \leq n_i} \tag{9}$$

Missing time points in the original time series $\vec{X}_i$ leads to missing values for corresponding pixels in the GADF image G_i; however, similar to the UTS case, we employ a simple linear interpolation on vectorized version of the image G_i to predict the value of missing pixels, which turned out to be efficient enough in our overall pipeline.

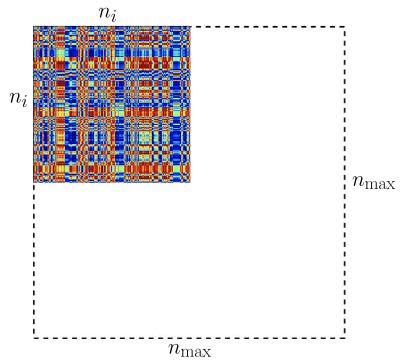


Fig. 3. Zero padding a $n_i \times n_i$ image to $n_{\max} \times n_{\max}$

Now that we have encoded the i^{th} component $\vec{X}_i$ of length n_i into an $n_i \times n_i$ image for all $1 \leq i \leq k$, we need to concatenate them vertically in the same order they appear in the MTS X; however, since these images have different sizes, we take the maximum size $n_{\max} = \max(n_1, n_2, ..., n_k)$ and zero pad each image to extend its shape into $n_{\max} \times n_{\max}$. This procedure is illustrated in Fig. 3. By vertically concatenation of these figures, we obtain one image of size $(kn_{\max}) \times n_{\max}$ corresponding to each MTS. Note that the way in which these images are concatenated (e.g. vertically or horizontally) does not matter as long as we are consistent across all labeled data because the filters in convolutional networks are spatially invariant.

Now that each MTS is converted to an image, same model as in Fig. 1 can be used again, i.e., the corresponding image for each MTS is passed through the Inception v3 model to create a new rich set of features. These features are then used to train a DNN for the classification task. While optimization over hyperparameters of the DNN can result in higher performance, we use the same structure as described earlier in section II.B. for univariate TSC to demonstrate its performance in the MTS setting. This procedure is summarized in the flowchart Algorithm 1.

C. Experiments & Results

First, we evaluate our classifier PD on five standard UCI benchmark MTS datasets and compare its efficiency to prior works. These datasets contain force and torque measurements on a robot after failure detection, and they define different learning problems for robot execution failure such as failures in motion with part, failures in approach to grasp position etc. Since a standard train and test split is not provided for these datasets, we need to follow the same experimental setup as prior works to make a fair comparison between our model and theirs. In Table II, the classifier PD is compared to 6 core classifiers, i.e., DTW, DDTW, DD_{DTW}, $1\text{NN}_{\text{MDDTW}}$, $\text{SVM}_{\text{MDDTW}}$ and C_{ADE}. Following the scheme of [6] and [7], a 10-fold cross-validation is employed

Algorithm 1: PD Multivariate Time Series Classifier

Data. *DATA*: A set of labeled MTS of dimension k.
Goal. Prediction on the set *TEST_SET* of MTS.
1. Initialize the set *DNN_DATA* $= \varnothing$.
2. for *MTS* $X = (\vec{X}_1, ..., \vec{X}_k) \in$ *DATA* **do**
  for $i \in \{1, 2, ..., k\}$ **do**
    – Transform $\vec{X}_i$ into the GADF image G_i.
    – Interpolate G_i for missing pixel values.
    – Zero pad G_i to the maximum image size.
  end
  – Concatenate all k images vertically.
  – Pass the concatenated image through Inception v3 to obtain a vector of length 2048.
  – Label this vector by the label of X in *DATA*.
  – Add this labeled vector to *DNN_DATA*.
end
3. Train the following DNN on *DNN_DATA*:
- 3 hidden layers with 800, 400 and 100 nodes in the 1st, 2nd and 3rd hidden layer respectively.
- ReLU activation function in hidden layers.
- Batch normalization after each activation layer.
- 50% dropout rate in all hidden layers.
- Softmax activation at the output layer.

4. for *MTS* $X = (\vec{X}_1, ..., \vec{X}_k) \in$ *TEST_SET* **do**
  for $i \in \{1, 2, ..., k\}$ **do**
    – Transform $\vec{X}_i$ into the GADF image G_i.
    – Interpolate G_i for missing pixel values.
    – Zero pad G_i to the maximum image size.
  end
  – Concatenate all k images vertically.
  – Pass the image through Inception v3.
  – Pass the output of Inception v3 through the DNN to predict the label for *MTS*.
end

where each dataset is partitioned randomly. Table II indicates that not only model PD outperforms other methods in 3 out of 5 datasets, but also has higher average accuracy. Similarly, following the experimental setup as in [16], a 5-fold cross-validation is employed to compare the performance of our method against four other core classifiers, SVM_{DTW}, 1NN_{WD}, TDVM and SMTS. Table III summarizes the results and shows that not only classifier PD outperforms other classifiers in four out of five datasets, but also leads to a higher average accuracy. Finally, following the train-test split specified in [3] we compare the classifier PD versus four deep learning based models, called LSTM-FCN, MLSTM-FCN, ALSTM-FCN and MALSTM-FCN in which a fully convolutional block and a LSTM block is used for multivariate TSC. Table IV summarizes the results and demonstrates that Classifier PD outperforms previous state-of-the-art deep learning models in four out of five datasets and also has a higher average accuracy.

Table II: 10-fold CV best classification accuracy rate (%)

	DTW	DDTW	DD-DTW	1NN-MDDTW	SVM-MDDTW	C_{ADE}	PD
LP1	87.36	77.5	85.14	90.9	92.1	**99.25**	94.13
LP2	68	62	68	72.3	67.9	77	**80**
LP3	71	71	75	70.2	74.5	77.14	**81.9**
LP4	89.92	79.55	89.92	92.3	96.6	**97.15**	95.1
LP5	70.7	62.68	71.25	72.5	71.3	72.65	**79.2**
Ave. Acc.	77.4	70.55	77.86	79.64	80.48	84.64	**86.07**

Table III: 5-fold CV best classification accuracy rate(%)

	SVM_{DTW}	1NN_{WD}	TDVM	SMTS	PD
LP1	81.8	81.8	85.2	**93.8**	92.47
LP2	63.8	59.6	63.8	61.6	**70.78**
LP3	65.8	61.7	68.1	81.1	**82.2**
LP4	87.2	86.3	85.5	93.7	**94.43**
LP5	62.1	65.2	67.1	73.9	**76.66**
Ave. Accuracy	72.14	70.92	73.94	80.82	**83.31**

Table IV: Accuracy rate(%) of PD vs LSTM-FCN models.

	LSTM-FCN	MLSTM-FCN	ALSTM-FCN	MALSTM-FCN	PD
LP1	74	86	80	82	**89.6**
LP2	77	**83**	80	77	73.3
LP3	67	80	80	73	**82**
LP4	91	92	89	93	**94**
LP5	61	66	62	67	**71.6**
Ave. Accuracy	74	81.4	78.2	78.4	**82.1**

In Table V, performance of the model PD is measured over four datasets in company PD. Due to privacy policy, these datasets are not publicly available; however, we provide a brief overview of the statistics for these data. Each dataset contains measurements over five metrics collected from every application running on the cluster; therefore, each instance is a multivariate time series of dimension five. Moreover, the length of each TS varies in a wide range between 10 and 10000. Since nodes may go down and not report any measurement for a long time, each TS can have up to 400 missing data points. The number of instances in training and test sets for one month worth of data as well as the number of labels for each dataset are specified in the first three column of Table V. The test accuracy of model PD is reported in the fourth column of Table V; however, since datasets are highly unbalanced, accuracy alone is not an appropriate evaluation metric for the performance of classifier. Hence, we also report the weighted precision, recall and F1-score in the subsequent columns. For each of the precision, recall and F1-score metrics, we calculate that metric for each label, and find their average weighted by support (the number of true instances for each label) that can result in an F1-score that is not between precision and recall but it accounts for label imbalance. Finally, accuracy of random chance prediction is included in the last column of

Table V: Performance of classifier PD over company PD datasets

	# Training Data	# Test Data	# Classes	Accuracy (%)	Precision	Recall	F1-Score	Random Guess (%)
PD Dataset G	226236	56540	10	95.80	0.9561	0.9580	0.9522	60.17
PD Dataset D	637904	159408	42	67.11	0.6654	0.6711	0.6469	20.24
PD Dataset NC	157428	39340	12	65.54	0.5990	0.6554	0.6113	47.04
PD Dataset NS	235176	58768	16	82.78	0.8253	0.8278	0.8075	53.89

Table V as a baseline. Comparison between the two reported accuracies reveals that in spite of having a large number of varied length MTS with missing data, classifier PD improves significantly upon the baseline in a tractable amount of time.

IV. Conclusion

We have presented Classifier PD for univariate and multivariate TSC. Our framework maps each TS to an image, extracts its features using a pre-trained CNN (Inception v3 model) and uses these features for training and prediction in a deep neural network. Our method scales linearly with the amount of data and outperforms previous works on UCR and UCI benchmark datasets. Finally, evaluation of the model PD on our private datasets in company PD demonstrates its applicability to large datasets, which is a crucial aspect of big data processing, and shows its robustness to variable length time series and missing data points.

References

[1] Z. Wang, W. Yan, and T. Oates, Time series classification from scratch with deep neural networks: A strong baseline, in Proc. Int. Joint Conf. Neural Netw. (IJCNN), May 2017, pp. 1578-1585.

[2] Z. Wang and T. Oates, Imaging time-series to improve classification and imputation, arXiv preprint arXiv:1506.00327, 2015.

[3] F. Karim, S. Majumdar, H. Darabi, and S. Harford, Multivariate LSTM-FCNs for Time Series Classification, arXiv preprint arXiv:1801.04503, 2018.

[4] P. Schäfer and U. Leser, Fast and accurate time series classification with weasel, In Proceedings of the 2017 ACM on Conference on Information and Knowledge Management (pp. 637-646), ACM, November 2017.

[5] A. Bostrom and A. Bagnall, Binary shapelet transform for multiclass time series classification, in Transactions on Large-Scale Data- and Knowledge-Centered Systems PDXII (Lecture Notes in Computer Science), vol. 10420, A. Hameurlain, J. Kng, R. Wagner, S. Madria, and T. Hara, Eds. Berlin, Germany: Springer, 2017.

[6] L. Wang, Z. Wang, and S. Liu, An effective multivariate time series classification approach using echo state network and adaptive differential evolution algorithm, Expert Syst. Appl., vol. 43, pp. 237-249, Jan. 2016.

[7] J. Mei, M. Liu, Y.-F. Wang, and H. Gao, Learning a Mahalanobis distance-based dynamic time warping measure for multivariate time series classification, IEEE Trans. Cybern., vol. 46, no. 6, pp. 1363-1374, Jun. 2016.

[8] Yanping Chen, et al, The UCR Time Series Classification Archive, 2015, URL www.cs.ucr.edu/ eamonn/time_series_data/.

[9] P. Schäfer, The boss is concerned with time series classification in the presence of noise, Data Mining and Knowledge Discovery, vol. 29, no. 6, pp. 1505-1530, 2015.

[10] P. Schäfer, Scalable time series classification, Data Mining and Knowledge Discovery, pp. 126, 2015.

[11] T. Górecki and M. Łuczak, "Multivariate time series classification with parametric derivative dynamic time warping," Expert Systems with Applications, vol. 42, pp. 2305-2312, 2015.

[12] J. Lines and A. Bagnall, Time series classification with ensembles of elastic distance measures, Data Mining Knowl. Discovery, vol. 29, no. 3, pp. 565-592, 2015.

[13] A. Bagnall, J. Lines, J. Hills, and A. Bostrom, Time-series classification with cote: the collective of transformation-based ensembles, IEEE Transactions on Knowledge and Data Engineering, vol. 27, no. 9, pp. 2522-2535, 2015.

[14] S. Spiegel, B.-J. Jain, and S. Albayrak. Fast time series classification under lucky time warping distance. In ACM SAC, pages 71-78, 2014.

[15] J. Grabocka, N. Schilling, M. Wistuba, and L. Schmidt-Thieme, Learning time-series shapelets, in Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, pp. 392-401, 2014.

[16] M. G. Baydogan, G. Runger, Learning a symbolic representation for multivariate time series classification, Data Mining and Knowledge Discovery, pp. 1-23, 2014.

[17] M. Baydogan, G. Runger, and E. Tuv, A bag-of-features framework to classify time series, IEEE Trans. Pattern Anal. Mach. Intell., vol. 25, no. 11, pp. 2796-2802, Jun. 2013.

[18] J. Lines, L. M. Davis, J. Hills, and A. Bagnall, A shapelet transform for time series classification, in Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 289-297, 2012.

[19] C. Orsenigo and C. Vercellis, Combining discrete SVM and fixed cardinality warping distances for multivariate time series classification, Pattern Recognit., vol. 43, no. 11, pp. 3787-3794, 2010.

[20] M. Vlachos, M. Hadjieleftheriou, D. Gunopoulos, and E. Keogh, Indexing Multi-Dimensional Time-Series with Support for Multiple Distance Measures, Proc. ACM SIGKDD, pp. 216-225, 2003.

[21] E. Keogh and C.A. Ratanamahatana, Exact Indexing of Dynamic Time Warping, Proc. 28th Int'l Conf. Very Large Data Bases, pp. 406-417, 2002.

[22] E. J. Keogh and M. J. Pazzani. Derivative Dynamic Time Warping. In SIAM SDM'01, 2001.

[23] H. Shimodaira, K.-I. Noma, M. Nakai, and S. Sagayama, Support vector machine with dynamic time-alignment kernel for speech recognition, in Proc. Eurospeech, vol. 3, pp. 18411844, 2001.

[24] R. Agrawal, K.-I. Lin, H.S. Sawhney, and K. Shim, "Fast Similarity Search in the Presence of Noise, Scaling, and Translation in Time-Series Databases," Proc. 21th Int'l Conf. Very Large Data Bases, pp. 490-501, Sept. 1995.

[25] D.M. Gavrila and L.S. Davis, Towards 3-d model-based tracking and recognition of human movement: a multi-view approach, In Proc. of the Int. Workshop on Automatic Face- and Gesture-Recognition, Zurich, pp. 272-277, 1995.