Multilayer Evolving Fuzzy Neural Networks with Self-Adaptive Dimensionality Compression for High-Dimensional Data Classification

Xiaowei Gu, Qiang Ni and Qiang Shen

***Abstract*—High-dimensional data classification is widely considered as a challenging task in machine learning due to the so-called "curse of dimensionality". In this paper, a novel multilayer jointly evolving and compressing fuzzy neural network (MECFNN) is proposed to learn highly compact multi-level latent representations from high-dimensional data. As a meta-level stacking ensemble system, each layer of MECFNN is based on a single jointly evolving and compressing neural fuzzy inference system (ECNFIS) that self-organises a set of human-interpretable fuzzy rules from input data in a sample-wise manner to perform approximate reasoning. ECNFISs associate a unique compressive projection matrix to each individual fuzzy rule to compress the consequent part into a tighter form, removing redundant information whilst boosting the diversity within the stacking ensemble. The compressive projection matrices of the cascading ECNFISs are self-updating to minimise the prediction errors via error backpropagation together with the consequent parameters, empowering MECFNN to learn more meaningful, discriminative representations from data at multiple levels of abstraction. An adaptive activation control scheme is further introduced in MECFNN to dynamically exclude less activated fuzzy rules, effectively reducing the computational complexity and fostering generalisation. Numerical examples on popular high-dimensional classification problems demonstrate the efficacy of MECFNN.**

***Index Terms*—high-dimensional data classification; compressive projection; dimensionality compression; evolving fuzzy system; stacking ensemble.**

I. Introduction

THANKS to the rapid advancements of electronic and manufacturing technologies, human societies have entered the era of big data. Massive volumes of digital data in the formats of texts, images, audio, videos, etc., are generated from different aspects of human activities over the past years with growing scale, complexity and dimensionality [1]. Mining, analysing and interpreting high-dimensional data is of great importance in various fields, including science, technology, healthcare and finance. Techniques capable of implementing these tasks play a crucial role in advancing knowledge, fostering innovation, and enabling informed decision-making. However, high dimensionality of data presents significant challenges to machine learning models due to the more complicated data structure, and raises many issues that can affect their abilities to accurately recognise the underlying data patterns and perform approximate reasoning. Learning from high-dimensional data is also computationally expensive and often leads to overfitting [2].

As a forefront of machine learning research, artificial neural networks (ANNs or deep neural networks, DNNs) have made remarkable achievements in a wide range of practical applications involving text, audio and visual information processing [3]. ANNs have demonstrated the state-of-the-art (SOTA) performances on many high-dimensional, complex problems that traditional machine learning models may struggle with [4]. The eye-catching successes of ANNs come from their capabilities to learn highly descriptive latent representations of the input data at multiple levels of abstraction through the utilisation of multiple layers of artificial neurons typically trained using backpropagation algorithms. Nevertheless, ANNs are extremely complicated systems composed of huge amounts of parameters preserving abstract information learned from data without clear physical meanings to be linked to the practical problems directly. As a result, there is often a lack of understanding on the rationale behind their decisions [5]. This also makes it practically impossible for human experts to identify the causes of prediction errors made by ANNs and fix these errors.

In addressing such observations, a number of post-hoc approaches have been introduced aiming to provide explanations to the internal reasoning of ANNs, e.g., saliency [6], layer-wise relevance propagation [7]. Yet, the vast majority of the resulting approaches focus on attributing the predictions of the ANNs to the inputs [8]. Their explanations are aligned to the modes' behaviours rather than human understanding, and may even be misleading and meaningless [5]. Despite of their superior prediction performances, concerns on the lack of interpretability and explainablity of ANNs have largely limited their wider adaption in real-world applications, particularly, for high-stake tasks.

Evolving fuzzy systems (EFSs) are a type of fuzzy systems that are capable of self-evolving the system structure and self-adjusting parameters online to capture the dynamically changing patterns of data streams, while summarising the captured information from data into a set of human-interpretable IF-THEN fuzzy rules [9], [10]. EFSs are an effective and prominent tool for real-time non-stationary problem approximation, offering high-level transparency and interpretability [11]. They can be implemented in the form of fuzzy rule-based systems

X. Gu is with the Department of Computer Science, University of Surrey, Guildford, GU2 7XH, UK. email: xiaowei.gu@surrey.ac.uk.

Q. Ni is with the School of Computing and Communications, Lancaster University, Lancaster, LA1 4WA, UK. email: q.ni@lancaster.ac.uk.

Q. Shen is with the Department of Computer Science, Aberystwyth University, Aberystwyth, SY23 3DB, UK. email: qqs@aber.ac.uk.

Corresponding author: Xiaowei Gu

Manuscript received XXXX XX, 2023; revised XXXX XX, 2023.

or neuro-fuzzy systems, and have been widely applied in many real-world applications for handling data streams with inherent uncertainties [12]. As one of the hottest topics in computational intelligence research, a wide variety of EFSs have been proposed over the past few decades.

The most representative examples of the recently developed EFSs include, but are not limited to, dynamic evolving neural-fuzzy inference system [9], evolving Takagi-Sugeno system [13], sequential adaptive fuzzy inference system [14], evolving fuzzy rule-based classifiers [10], evolving granular neural network [15], parsimonious network based on fuzzy inference system [16], generic evolving neuro-fuzzy inference system [17], evolving fuzzy model [18], self-evolving fuzzy system [19], parsimonious learning machine [71], evolving fuzzy system with self-learning/adaptive thresholds [21], jointly evolving and compressing fuzzy system [22], statistically evolving fuzzy inference system (SEFIS) [23], etc. With the transparent, highly flexible system structure and human-interpretable reasoning mechanism, EFSs have demonstrated great success in handling real-world streaming data problems, providing an effective and promising solution toward explainable artificial intelligence [24].

Although highly complex models do not necessarily achieve greater prediction accuracy, it is generally recognised that the simpler and more transparent EFSs struggle to handle high-dimensional and complex problems, say for image recognition, due to the so-called "curse of dimensionality" [22]. Existing EFSs suffer from system obesity with significantly increased complexity and lose transparency when applied to such problems. This is because of the inherent vulnerability of fuzzy systems to the curse of dimensionality [25], [26]. In practice, as the dimensionality of data increases, both the number of fuzzy rules and the number of parameters associated with the rules increase significantly, which not only impairs the interpretability, but also leads to system breakdown potentially. One feasible solution to tackle system obesity is to cap the number of fuzzy rules within the system [27]. However, this approach carries a risk that the learned fuzzy rules may be insufficient for precisely approximating the problem. In addition, EFSs rely on online clustering for fuzzy rule identification from incoming data. Unfortunately, the validity and reliability of the obtained clusters unfavourably decrease with the increase of input dimensionality. As a result, EFS models learned from high-dimensional data are prone to be overfitting and the prediction performances are usually limited. To empower the capability of EFSs for handling sophisticated problems, many flat ensemble schemes have been proposed to construct stronger predictive models from a number of individual single-model EFSs, trained either in parallel [28]–[33] or sequentially [34], [35]. The performance improvement of flat ensemble schemes is typically attained by boosting the diversity among the individual EFSs and increasing the width of the resulting ensemble model. However, this is usually insufficient to tackle high-dimensional, complicated learning tasks. To gain greater representation learning capabilities, the depth of the ensemble model has to be increased [36], a lesson learned from the conventional ANN paradigm also [37].

To date, there has been a growing number of works in the literature attempting to build multilayer stacking ensemble models with EFSs to learn multi-level latent representations from data. In [38], a deep evolving fuzzy neural network (DEVFNN) is proposed. It is composed of multiple cascading EFSs (one per layer) with the outputs of EFSs at previous layers being utilised to augment the inputs to the EFSs at successive layers. A multilayer ensemble evolving fuzzy inference system (MEEFIS) is introduced in [39]. Each layer of MEEFIS is an ensemble of multiple EFSs implemented in parallel to jointly process the inputs received from preceding layer. These early techniques explore the potential of EFSs in stacked ensemble learning. Both methods utilise the standard recursive least squares (RLS)-based algorithms [9], [10] to adjust the consequent parameters of the individual EFSs within the ensemble models. However, consequent parameter learning using RLS can be computationally expensive when the dimensionality of data is high due to the recursive updating of covariance matrices. In [40], a hierarchical evolving fuzzy system (HEFS) that employs the kernel conjugate gradient (KCG) algorithm for training consequent parameters is proposed. While KCG offers greater computational efficiency compared to RLS when dealing with low-dimensional data, its computational complexity also increases substantially as the dimensionality of data increases.

In addition, one significant issue overlooked by the vast majority of existing works on stacking ensemble EFS models is that conventional consequent parameter learning algorithms, such as RLS and KCG, require the error functions to be explicitly specified. As such, given a current input individual, EFSs have to be trained separately to minimise the difference between their outputs and the ground truth, in order to avoid ambiguity in defining the error functions for different layers. This updating mechanism limits the interaction between EFSs at different layers and hinders multi-level representation learning [41]. Having recognised the limitation of conventional algorithms in stacking ensemble consequent parameter learning, recent designs [41], [42] tend to employ back propagation instead for adjusting the parameters of ensemble components layer-by-layer. This helps break the bottleneck, fostering the interactions between layers to learn more descriptive latent representations from data collaboratively. Despite such advancement, learning stacking EFS ensembles from high-dimensional, complicated problems remains a highly challenging task due to the potentially redundant information in the data, hindering effective generalisation [22].

Compressive sensing techniques, such as very sparse random projection (VSRP) [43], have been proven useful for EFSs to learn a more compact fuzzy rule base from high-dimensional data, improving computational efficiency and reducing overfitting [22], [33]. Very recently, a multilayer stacked evolving fuzzy system (MS-EFS) combined with dimensionality compression is proposed in [44]. In MS-EFS, each individual EFS is followed by a feature compressing layer based on VSRP, which compresses the dimensionality of its outputs before passing them as inputs to the successive EFS. While the computational complexity of the first EFS in MS-EFS remains high due to the use of weighted RLS algorithm for consequent parameter learning, the VSRP matrices

effectively improve computational efficiency of the subsequent EFSs in the stacking ensemble and increase the compactness of the learned IF-THEN rules. Nonetheless, in practice, utilising VSRP matrices in EFSs inevitably impairs the prediction performance of the model due to the loss of information after data compression, particularly when the compression ratio is high [33]. The primary reason for this degradation is that these VSRP matrices are randomly generated during the system identification process and are not adapted to the input data. This random generation process does not account for the specific characteristics of the data, leading to potential loss of important information and, consequently, a decline in prediction accuracy.

To better handle high-dimensional complex problems, the following three limitations of existing EFS-based stacking ensemble models need to be addressed collectively: 1) insufficient interaction between different layers; 2) significant increase in computational complexity due to high dimensionality; and 3) inefficiency in eliminating redundant information from the data. In this paper, a novel Multilayer jointly Evolving and Compressing Fuzzy Neural Network (MECFNN) that overcomes these three important limitations is proposed for high-dimensional data classification. MECFNN is a multilayer stacking ensemble model composed of multiple jointly Evolving and Compressing Neural Fuzzy Inference Systems (ECNFISs) arranged in layers to learn more meaningful and compact latent representations from data at multiple levels of abstraction. It works through the utilisation of self-adaptive compressive projection (SACP) matrices for autonomous dimensionality compression, which cleans redundant information in data and further enhances the diversity between the learned IF-THEN fuzzy rules as well as between the learned network layers. Similar to its predecessor, multilayer evolving fuzzy neural network (MEFNN) [41], each layer of MECFNN is based on a single ECNFIS self-organising and self-evolving from data on a sample-by-sample basis. Inputs to MECFNN are processed by the cascading ECNFISs layer-by-layer to produce the final outputs, and the prediction errors are feed backwards for parameter updating.

A distinctive feature of MECFNN is that the consequent part of each individual IF-THEN fuzzy rule is associated with a unique SACP matrix. These SACP matrices are continuously self-updating with data to minimise the prediction errors. In doing so, MECFNN can gradually learn to preserve the more discriminative features during dimensionality compression and to discard the less relevant ones, rather than in a purely random manner as done with the existing techniques [22], [33], [44]. To reduce the computational complexity and further improve generalisation, an adaptive activation control scheme is also introduced in MECFNN. It selects only these activated fuzzy rules by the current input for output generation and consequent parameter updating at each learning cycle [11], resembling dropout regularisation in ANNs. The utilisation of activation functions in dimensionality compression and output generation enables MECFNN to capture and reflect the subtle nonlinearity hidden in the data [41]. Hence, MECFNN possesses greater prediction performance when dealing with high-dimensional, complex problems.

To summarise, novel features of the proposed MECFNN include:

1) A meta-level stacking ensemble system composed of multiple cascading EFS models to self-evolve from data in a sample-wise manner and to simultaneously perform dimensionality compression, removing redundant information while boosting generalisation capacity;
2) An adaptive dimensionality compression scheme to promote the diversity within the stacking ensemble, enabling the model to learn more meaningful, compact multi-level latent representations from data through the exploitation of SACP matrices;
3) An adaptive activation control scheme to dynamically exclude less activated IF-THEN rules at different layers in response to the inputs in a more targeted, flexible manner, effectively reducing the computational complexity involved in updating the consequent part while preventing overfitting.

The proposed MECFNN is distinctive from existing techniques in the following five aspects:

1) In contrast to ANNs [3], [4], MECFNN self-organises a multilayer fuzzy rule-based system structure from data, offering greater model transparency and inference interpretability;
2) Different from EFS-based flat ensemble models [28]–[35], MECFNN learns latent representations from data at multiple levels of abstraction thanks to its deep structure;
3) Unlike most of EFS-based multilayer ensemble models [38]–[40], [44], MECFNN fosters information exchange between different layers using error backpropagation for consequent parameter learning;
4) Differing from alternative EFS models using dimensionality compression [22], [33], [44], MECFNN self-adapts the compressive projection matrices from data, better preserving more discriminative features;
5) Compared to its predecessor [41], MECFNN learns more meaningful, compact representations from data with improved generalisation with adaptive dimensionality compression and activation control.

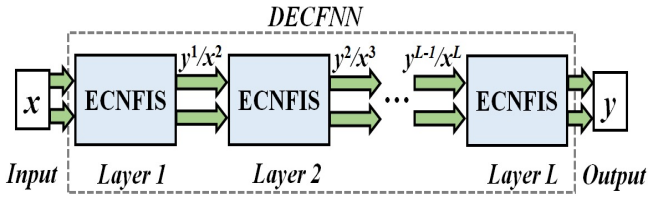
Numerical examples based on a range of high-dimensional benchmark classification problems demonstrate that with the substantial enhancements, MECFNN achieves superior representation learning capabilities in comparison to its predecessor, MEFNN and surpasses the SOTA approaches with elevated classification accuracy.

The remainder of this paper is organised as follows. Technical details of the proposed MECFNN are presented in Section II. Numerical examples on high-dimensional benchmark classification problems are given in Section III for comparative performance demonstration. This paper is concluded in Section IV, which also outlines directions for future work.

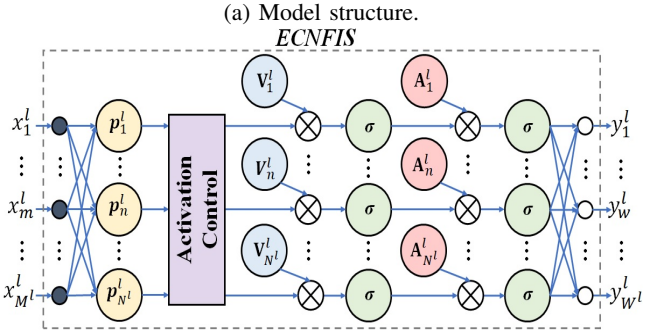
II. Proposed MECFNN

A. General Architecture

The model structure of MECFNN is depicted in Fig. 1a, where the zoom-in structure of the l^{th} layer is given by Fig. 1b. It can be seen from Fig. 1 that similar to its predecessor,



(a) Model structure.



(b) Inner structure of the l^{th} layer.

Fig. 1: General Architecture of MECFNN.

MEFNN [41], MECFNN is a meta-level ensemble system composed of multiple ECNFISs stacked on top of each other layer-by-layer, where the outputs of the preceding layer serve as the inputs of the following layer. Note that, different from existing EFSs in the literature, ECNFIS is equipped with two unique schemes, namely, *i)* adaptive dimensionality compression and *ii)* adaptive activation control [11].

Assuming MECFNN is composed of L individual ECNFISs, each ECNFIS (e.g., the l^{th} one) is a MIMO EFS (see Fig. 1b) composed of N^l prototype-based IF-THEN rules in the form of Eq. (1) [33], [41].

$$\boldsymbol{R}_n^l: \quad IF\ (\boldsymbol{x}^l \sim \boldsymbol{p}_n^l)\ \ THEN\ (\boldsymbol{y}_n^l = \sigma(\mathbf{A}_n^l[1, \boldsymbol{c}_n^l]^T)) \quad (1)$$

where $l = 1, 2, ..., L$; $\boldsymbol{R}_n^l$ denotes the n^{th} IF-THEN rule of the l^{th} ECNFIS; $n = 1, 2, ..., N^l$; "$\sim$" denotes similarity; $\boldsymbol{x}^l = [x_1^l, x_2^l, ..., x_{M^l}^l]^T$ is the $M^l \times 1$ dimensional input vector; $\boldsymbol{y}_n^l = [y_{n,1}^l, y_{n,2}^l, ..., y_{n,W^l}^l]^T$ is the $W^l \times 1$ dimensional output of $\boldsymbol{R}_n^l$ in response to $\boldsymbol{x}^l$; $\boldsymbol{p}_n^l$ is the $M^l \times 1$ dimensional prototype/antecedent part of $\boldsymbol{R}_n^l$; $\boldsymbol{c}_n^l = \sigma^T(\mathbf{V}_n^l \boldsymbol{x}^l)$; $\mathbf{V}_n^l = [v_{n,j,k}^l]_{k=1:M^l}^{j=1:C^l}$ is the $C^l \times M^l$ dimensional SACP matrix associated with $\boldsymbol{R}_n^l$; $C^l = \epsilon^l M^l$; ϵ^l is the compression ratio of the l^{th} ECNFIS, $0 < \epsilon^l \leq 1$; $\mathbf{A}_n^l = [a_{n,j,k}^l]_{k=0:C^l}^{j=1:W^l}$ is the corresponding $W^l \times (C^l + 1)$ dimensional consequent parameter matrix; and $\sigma(\cdot)$ denotes the standard activation function. In this study, the classic sigmoid function is employed, namely, $\sigma(x) = \frac{1}{1+e^{-x}}$.

One distinctive feature of MECFNN is that the consequent part of each IF-THEN rule is associated with a unique SACP matrix, $\mathbf{V}_n^l$ that maps the input vectors onto a lower dimensional latent space. Upon identifying and adding a new IF-THEN rule to the rule base, a newly produced SACP matrix is assigned to its consequent part in the form of a randomly generated VSRP matrix (as specified by Eq. (5)). After initialisation, such SACP matrices will be continuously adjusted during the learning process through error backpropagation to better preserve the key information from data whilst reducing redundant information (as specified by Eq. (25)). Since every SACP matrix is unique, the input data to a ECNFIS is projected onto different latent spaces, offering multiple views of the underlying data. This uniqueness property effectively enhances the diversity within the rule bases of individual ECNFISs.

The output, $\boldsymbol{y}^l$ of the l^{th} ECNFIS given the input $\boldsymbol{x}^l$ is derived by Eq. (2):

$$\boldsymbol{y}^l = f^l(\boldsymbol{x}^l) = \sum_{n=1}^{N^l} \lambda_n^l \boldsymbol{y}_n^l = \sum_{n=1}^{N^l} \lambda_n^l \sigma(\mathbf{A}_n^l[1, \boldsymbol{c}_n^l]^T) \quad (2)$$

where λ_n^l is the normalised firing strength of $\boldsymbol{R}_n^l$ (specified by Eq. (15) later).

Since the inputs of the l^{th} ECNFIS are the outputs of the $l-1^{th}$ ECNFIS and its outputs serve as the inputs of the $l+1^{th}$ ECNFIS, there are $\boldsymbol{x}^l = \boldsymbol{y}^{l-1}$ and $\boldsymbol{x}^{l+1} = \boldsymbol{y}^l$. Hence, IF-THEN fuzzy rules in the successive layers are fully connected, and the L-layer MECFNN can be modelled by the following composite function [41]:

$$\boldsymbol{y} = f^L \circ f^{L-1} \circ \cdots \circ f^2 \circ f^1(\boldsymbol{x}) \quad (3)$$

where $\boldsymbol{x} = [x_1, x_2, ..., x_M]^T$ and $\boldsymbol{y} = [y_1, y_2, ..., y_W]^T$ denote the input and output of MECFNN, respectively; y_j represents the likelihood that $\boldsymbol{x}$ belongs to the j^{th} class ($j = 1, 2, ..., W$).

Similar to MEFNN [41], users need to specify the output sizes (namely, W^1,W^2,...,W^{L-1}) of the stacked ECNFISs (except for the final one) and the corresponding compression ratios (namely, ϵ^1,ϵ^2,...,ϵ^L) a priori for MECFNN. It will self-organise its multi-level model structure and parameters automatically from input data based on their ensemble properties and mutual distances. The output sizes (W^l) control the amounts of information transmitting between layers, and the compression ratios (ϵ^l) control the dimensionality of the input vectors at the respective layers after compression. It is worth noting that both the output sizes and the compression ratios are not problem- and user- specific, and can be determined based on users' preference without prior knowledge of the problem or making any assumptions on the data generation model. A detailed discussion on the parameter settings of MECFNN is presented in Section III. A and the recommended setting is also given in the same section.

Thanks to the adaptive dimensionality compression scheme, MECFNN is capable of compressing the consequent parts of the IF-THEN rules into a tighter and more meaningful form to remove the redundant information whilst self-evolving from data simultaneously. This helps MECFNN to effectively capture the hierarchically interconnected patterns of data across different levels of abstraction and learn more compact, descriptive representations, lifting the "curse of dimensionality". Moreover, the adaptive activation control scheme works similarly as the dropout technique used by ANNs but in a more selective, flexible manner by dynamically dropping out these less activated IF-THEN rules with respect to the current inputs, effectively improving the computational efficiency of MECFNN and reducing overfitting [11].

In the next subsection, the system identification process of MECFNN is detailed. Note that all the ECNFISs forming the

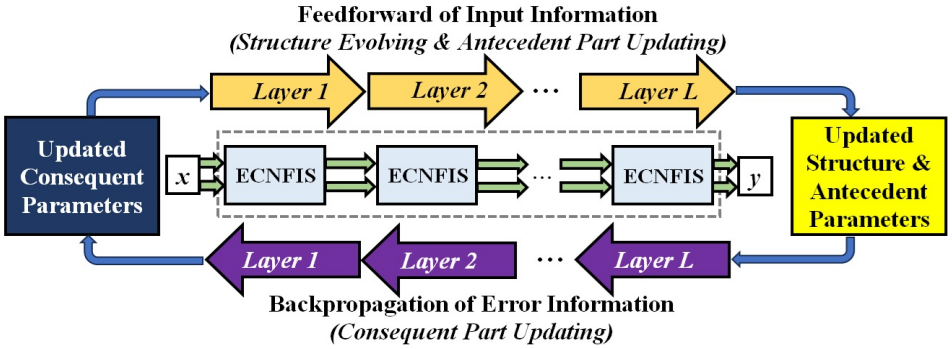


Fig. 2: Illustration of structure evolving and parameter updating process.

stacking ensemble follow the exact same structure evolving and parameter updating process during system identification.

B. System Identification

MECFNN self-organises the meta-level model structure and parameters automatically from data on a sample-by-sample basis and performs dimensionality compression simultaneously. The system identification process of MECFNN is composed of the following three stages. For clarity, the structure evolving and parameter updating process is illustrated in Fig. 3.

Stage 0. System Initialisation. For a particular input sample, $\boldsymbol{x}_k$ (k stands for the time instance at which $\boldsymbol{x}_k$ is observed), if $k = 1$, the meta-level structure and parameters of MECFNN will be initialised by $\boldsymbol{x}_k$ in a layer-wise manner starting from the ECNFIS at the first layer ($l = 1$). Otherwise, the system identification process skips Stage 0 and enters Stage 1 directly.

Given the first input sample, $\boldsymbol{x}_k^l$ ($\boldsymbol{x}_k^l = \boldsymbol{x}$ if $l = 1$ or $\boldsymbol{x}_k^l = \boldsymbol{y}_k^{l-1}$ $\forall l > 1$), the global parameters of the l^{th} ECNFIS are initialised as [11]:

$$\boldsymbol{\eta}^l \leftarrow \boldsymbol{x}_k^l; \quad X^l \leftarrow ||\boldsymbol{x}_k^l||^2 \tag{4}$$

where $\boldsymbol{\eta}^l$ and X^l denote the global means of all the input samples to the l^{th} ECNFIS and their squared Euclidean norms, respectively.

The first fuzzy rule, $\boldsymbol{R}_{N^l}^l$ ($N^l \leftarrow 1$) of the l^{th} ECNFIS is initialised in the form of Eq. (1) with the corresponding prototype $\boldsymbol{p}_{N^l}^l$, SACP matrix $\mathbf{V}_{N^l}^l$ and consequent parameter matrix $\mathbf{A}_{N^l}^l$ set by Eq. (5) [22], [33], [41].

$$\boldsymbol{p}_{N^l}^l \leftarrow \boldsymbol{x}_k^l; \quad \mathbf{V}_{N^l}^l \leftarrow \boldsymbol{\omega}_o^l; \quad \mathbf{A}_{N^l}^l \leftarrow \frac{1}{M^l+1}\boldsymbol{\xi}_o \tag{5}$$

where $\alpha^l = \sqrt{M^l}$; $\boldsymbol{\omega}_o^l = [\frac{\sqrt{\alpha^l}}{\sqrt{C^l}}\omega_{o,j,k}^l]_{k=1:M^l}^{j=1:C^l}$ is a unique $C^l \times M^l$ dimensional VSRP matrix with the elements randomly generated by the distribution given in Eq. (6) [22], [43]; $\boldsymbol{\xi}_o = [\xi_{o,j,k}]_{k=0:C^l}^{j=1:W^l}$ is a randomly generated $W^l \times (C^l + 1)$ dimensional matrix, whose elements equal to either 0 or 1, namely, $\xi_{o,j,k} \in \{0, 1\}$ $\forall$ j, k, following the symmetrical binomial distribution [41]. Note that, both $\mathbf{V}_{N^l}^l$ and $\mathbf{A}_{N^l}^l$ are randomly generated for each IF-THEN rule and will be continuously updated throughout the learning process to better approximate the data.

$$\omega_{o,j,k}^l = \begin{cases} 1, & \text{with probability } \frac{1}{2\alpha^l} \\ 0, & \text{with probability } 1 - \frac{1}{\alpha^l} \\ -1, & \text{with probability } \frac{1}{2\alpha^l} \end{cases} \tag{6}$$

The main reason for using VSRP matrices as the initial SACP matrices is that VSRP matrices can effectively reduce the dimensionality of data, improve the computational efficiency with little impact on accuracy (in the expectation) [22], [43]. However, in practice, the utilisation of VSRP matrices inevitably decreases the accuracy of the predictive model because these randomly generated matrices are often far from the optimum and can cause a significant loss of information after data compression [33]. Therefore, these matrices need to be continuously updated with data during the system identification process to achieve more meaningful compression.

Then, parameters of the cluster, denoted as $\boldsymbol{C}_{N^l}^l$ associated with $\boldsymbol{R}_{N^l}^l$ are initialised by Eq. (7) [41].

$$\boldsymbol{m}_{N^l}^l \leftarrow \boldsymbol{x}_k^l; \quad \chi_{N^l}^l \leftarrow ||\boldsymbol{x}_k^l||^2; \quad S_{N^l}^l \leftarrow 1 \tag{7}$$

where $\boldsymbol{m}_{N^l}^l$ and $\chi_{N^l}^l$ are the respective arithmetic means of all data samples associated with $\boldsymbol{C}_{N^l}^l$ and their squared Euclidean norms; $S_{N^l}^l$ is the number of such samples, namely, support.

After the l^{th} ECNFIS is initialised, it produces the output y_k^l using Eq. (8) and passes it to the ECNFIS at the next layer ($l + 1^{th}$) as the input.

$$\boldsymbol{y}_k^l = \sigma(\mathbf{A}_{N^l}^l[1, \boldsymbol{c}_{N^l,k}^l]^T) \tag{8}$$

where $\boldsymbol{c}_{N^l,k}^l = \sigma^T(\mathbf{V}_{N^l}^l \boldsymbol{x}_k^l)$.

Once the $l + 1^{th}$ ECNFIS receives the input $\boldsymbol{x}_k^{l+1}$ from the previous layer ($\boldsymbol{x}_k^{l+1} \leftarrow \boldsymbol{y}_k^l$), the same initialisation process given by Eqs. (4) to (8) is repeated until the ECNFIS at the final layer (L^{th}) is initialised and the output of MECFNN is produced ($\boldsymbol{y}_k \leftarrow \boldsymbol{y}_k^L$). After this, the current learning cycle enters Stage 2 for consequent part updating.

Stage 1. Structure Evolving and Antecedent Part Updating. Given a new input sample $\boldsymbol{x}_k$ with $k > 1$, the model structure and antecedent parameters of MECFNN are adjusted accordingly to reflect the changes in the data patterns on a layer-wise basis by updating the cascading ECNFISs sequentially in response to the inputs.

With the current input sample $\boldsymbol{x}_k^l$ ($\boldsymbol{x}_k^l \leftarrow \boldsymbol{x}_k$ if $l = 1$ or $\boldsymbol{x}_k^l \leftarrow \boldsymbol{y}_k^{l-1}$, otherwise), global parameters of the l^{th} ECNFIS are firstly updated as ($l = 1, 2, ..., L$) [41]:

$$\boldsymbol{\eta}^l \leftarrow \boldsymbol{\eta}^l + \frac{\boldsymbol{x}_k^l - \boldsymbol{\eta}^l}{k}; \quad X^l \leftarrow X^l + \frac{||\boldsymbol{x}_k^l||^2 - X^l}{k} \tag{9}$$

Then, the firing strength of each IF-THEN rule, $\boldsymbol{R}_n^l$ ($n = 1, 2, ..., N^l$) in response to $\boldsymbol{x}_k^l$ is calculated using Eq. (10) [41].

$$\mu_n(\boldsymbol{x}_k^l) = e^{-\frac{||\boldsymbol{x}_k^l - \boldsymbol{p}_n^l||^2}{(\tau_n^l)^2}} \tag{10}$$

where $\tau_n^l = \sqrt{\frac{X^l - ||\boldsymbol{\eta}^l||^2 + \chi_n^l - ||\boldsymbol{m}_n^l||^2}{2}}$.

Condition 1 is then utilised to examine whether $\boldsymbol{x}_k^l$ represents a data pattern unseen from previous inputs [41]:

$$\begin{aligned} \text{Cond. 1}: \ &\text{if } \big(\max_{n=1,2,...,N^l}(\mu_n(\boldsymbol{x}_k^l)) < \delta_o\big) \\ &\text{then } (\boldsymbol{x}_k^l \text{ becomes a new prototype}) \end{aligned} \tag{11}$$

where δ_o ($0 < \delta_o < 1$) is the threshold to identify whether $\boldsymbol{x}_k^l$ is spatially distant to all existing prototypes and, by default, $\delta_o = e^{-3}$ is used.

If $\boldsymbol{x}_k^l$ meets Condition 1, it is suggested that $\boldsymbol{x}_k^l$ is distinctive from prototypes identified previously and highly likely to represent a data pattern unseen before. To capture this new data pattern, a new IF-THEN rule, $\boldsymbol{R}_{N^l}^l$ ($N^l \leftarrow N^l + 1$) and the associated cluster, $\boldsymbol{C}_{N^l}^l$ are initialised with the related parameters set by Eqs. (5)-(7). Otherwise, namely, Condition 1 is not satisfied, $\boldsymbol{x}_k^l$ is utilised for updating the parameters of the cluster, $\boldsymbol{C}_{n^*}^l$ associated with the IF-THEN rule producing the greatest firing strength ($n^* = \arg\max_{n=1,2,\ldots,n^*}(\mu_n(\boldsymbol{x}_k^l))$) by Eq. (12).

$$\begin{aligned} \boldsymbol{m}_{n^*}^l &\leftarrow \boldsymbol{m}_{n^*}^l + \frac{\boldsymbol{x}_k^l - \boldsymbol{m}_{n^*}^l}{S_{n^*}^l + 1} \\ \chi_{n^*}^l &\leftarrow \chi_{n^*}^l + \frac{||\boldsymbol{x}_k^l||^2 - \chi_{n^*}^l}{S_{n^*}^l + 1} \\ S_{n^*}^l &\leftarrow S_{n^*}^l + 1 \end{aligned} \tag{12}$$

After the model structure and antecedent parameters of the l^{th} ECNFIS are updated, the firing strength values of the IF-THEN rules within the rule base are recalculated using Eq. (10) to reflect the latest changes in respect to $\boldsymbol{x}_k^l$. Then, adaptive activation control is performed to enable ECNFIS to identify these more activated rules by the current input for parameter updating and output generation. To do so, the firing strength values are ranked in descending order, re-denoted by $\mu_1^*(\boldsymbol{x}_k^l)$, $\mu_2^*(\boldsymbol{x}_k^l)$,...,$\mu_{N^l}^*(\boldsymbol{x}_k^l)$ ($\mu_1^*(\boldsymbol{x}_k^l) \geq \mu_2^*(\boldsymbol{x}_k^l) \geq ... \geq \mu_{N^l}^*(\boldsymbol{x}_k^l)$) and the number of activated IF-THEN rules in the rule base with higher firing strength values than the rest is identified by Eq. (13).

$$\hat{N}_k^l = \underset{n=1,2,...,N^l}{\arg\min} \Big(\sum_{i=1}^{n} \mu_n^*(\boldsymbol{x}_k^l) \geq \rho_o \sum_{i=1}^{N^l} \mu_n(\boldsymbol{x}_k^l)\Big) \tag{13}$$

where $0 \leq \rho_o \leq 1$. Eq.(13) identifies the minimum number, $\hat{N}_k^l$ of such IF-THEN rules with the sum of their firing strength values exceeds the soft threshold specified by $\rho_o \sum_{i=1}^{N^l} \mu_n(\boldsymbol{x}_k^l)$. Hence, one can see that with $\rho_o = 1$, all the IF-THEN rules are considered as being activated, and; only a single rule giving the highest firing strength will be selected if ρ_o is set to be 0. In this study, $\rho_o = 0.95$ is used by default such that a small number of the least activated IF-THEN rules are excluded from output generation and consequent parameter updating (in Stage 2), effectively improving the computational efficiency and preventing overfitting.

Based on Eq. (13), Condition 2 is utilised to examine the IF-THEN rules one-by-one ($n = 1, 2, ..., N^l$) to see whether they are activated by the current input $\boldsymbol{x}_k^l$.

$$\begin{aligned} \text{Cond. 2}: \ &\text{if } \big(\mu_n(\boldsymbol{x}_k^l) \geq \mu_{\hat{N}_k^l}^*(\boldsymbol{x}_k^l)\big) \\ &\text{then } (\boldsymbol{R}_n^l \text{ is activated by } \boldsymbol{x}_k^l) \end{aligned} \tag{14}$$

The normalised firing strength values of the IF-THEN rules are derived by Eq. (15):

$$\lambda_{n,k}^l = \begin{cases} \frac{\mu_n(\boldsymbol{x}_k^l)}{\sum_{i=1}^{\hat{N}_k^l} \mu_i^*(\boldsymbol{x}_k^l)}, & \boldsymbol{R}_n^l \text{ meets Cond. 2} \\ 0, & \text{otherwise} \end{cases} \tag{15}$$

Then, the output of the l^{th} ECNFIS, $\boldsymbol{y}_k^l$ with regards to $\boldsymbol{x}_k^l$ is generated using Eq. (2), and passed to the $l+1^{th}$ ECNFIS at the next layer as the input. The same process (Eqs. (9)-(15)) is repeated until the model structure and antecedent parameters of L^{th} ECNFIS at the final layer are updated and the output of MECFNN, $\boldsymbol{y}_k$ ($\boldsymbol{y}_k \leftarrow \boldsymbol{y}_k^L$) is produced. After this, the system identification process enters Stage 2 to update the consequent parts of the cascading ECNFISs, which include the consequent parameter matrices $\mathbf{A}_n^l$ and SACP matrices $\mathbf{V}_n^l$ ($\forall l, n$).

Stage 2. Consequent Part Updating. Same as MEFNN [41], MECFNN utilises backpropagation for consequent parameter updating for the following three reasons: *i)* avoiding the ambiguity in specifying the error functions for updating the consequent parameters of individual ECNFISs at different layers as required by standard RLS-based algorithms used by conventional EFSs; *ii)* encouraging information exchange between layers to collaboratively learn more descriptive, compact multi-level representations from data after dimensionality compression, and; *iii)* removing the need of updating covariance matrices for consequent parameter learning to achieve greater computational efficiency, particularly, in high-dimensional problems.

Given the final output $\boldsymbol{y}_k$, the prediction error is defined as:

$$\boldsymbol{e}_k = \frac{1}{2}(\boldsymbol{y}_k - \boldsymbol{r}_k)^T(\boldsymbol{y}_k - \boldsymbol{r}_k) \tag{16}$$

where $\boldsymbol{r}_k = [r_{k,1}, r_{k,2}, \ldots, r_{k,W}]^T$ is the corresponding one-hot encoded class label of $\boldsymbol{x}_k$ [41].

Consequently, the derivative of $\boldsymbol{e}_k$ with respect to $\boldsymbol{y}_k^L$ is given by Eq. (17):

$$\frac{\partial \boldsymbol{e}_k}{\partial \boldsymbol{y}_k^L} = \boldsymbol{y}_k^L - \boldsymbol{r}_k \tag{17}$$

Based on Eq. (17), the consequent parameter matrices and SACP matrices of the IF-THEN rules of the cascading ECNFISs are updated layer-by-layer in a backward manner from the last layer to the first layer. The derivatives of the two

matrices in the consequent part of the n^{th} IF-THEN rule of the l^{th} ECNFIS are given as follows [41] ($n = 1, 2, ..., N^l$).

$$\frac{\partial \boldsymbol{e}_k}{\partial \mathbf{A}_n^l} = \frac{\partial \boldsymbol{e}_k}{\partial \boldsymbol{y}_k^L} \cdot \frac{\partial \boldsymbol{y}_k^L}{\partial \boldsymbol{y}_k^{L-1}} \cdot \cdots \cdot \frac{\partial \boldsymbol{y}_k^{l+1}}{\partial \boldsymbol{y}_k^l} \cdot \frac{\partial \boldsymbol{y}_k^l}{\partial \mathbf{A}_n^l} = \boldsymbol{d}_k^l \cdot \frac{\partial \boldsymbol{y}_k^l}{\partial \mathbf{A}_n^l} \quad (18)$$

$$\frac{\partial \boldsymbol{e}_k}{\partial \mathbf{V}_n^l} = \frac{\partial \boldsymbol{e}_k}{\partial \boldsymbol{y}_k^L} \cdot \frac{\partial \boldsymbol{y}_k^L}{\partial \boldsymbol{y}_k^{L-1}} \cdot \cdots \cdot \frac{\partial \boldsymbol{y}_k^{l+1}}{\partial \boldsymbol{y}_k^l} \cdot \frac{\partial \boldsymbol{y}_k^l}{\partial \mathbf{V}_n^l} = \boldsymbol{d}_k^l \cdot \frac{\partial \boldsymbol{y}_k^l}{\partial \mathbf{V}_n^l} \quad (19)$$

where $\boldsymbol{d}_k^l = \frac{\partial \boldsymbol{e}_k}{\partial \boldsymbol{y}_k^L} \cdot \frac{\partial \boldsymbol{y}_k^L}{\partial \boldsymbol{y}_k^{L-1}} \cdot \cdots \cdot \frac{\partial \boldsymbol{y}_k^{l+1}}{\partial \boldsymbol{y}_k^l}$ is the derivative of prediction error with respect to $\boldsymbol{y}_k^l$ and there is $\boldsymbol{d}_k^L = \frac{\partial \boldsymbol{e}_k}{\partial \boldsymbol{y}_k^L}$.

Utilising the chain rule, $\boldsymbol{d}_k^{l-1}$ can be derived from $\boldsymbol{d}_k^l$ using Eq. (20) ($\forall l = 2, 3, ..., L$):

$$\begin{aligned} \boldsymbol{d}_k^{l-1} = \boldsymbol{d}_k^l \cdot \frac{\partial \boldsymbol{y}_k^l}{\partial \boldsymbol{x}_k^l} = \sum_{n=1}^{N^l} \big(& \frac{\partial \lambda_{n,k}^l}{\partial \boldsymbol{x}_k^l} \cdot (\boldsymbol{y}_{n,k}^l)^T \cdot \boldsymbol{d}_k^l \\ & + \lambda_{n,k}^l \cdot (\tilde{\mathbf{A}}_n^l \mathbf{U}_{n,k}^l)^T \cdot \boldsymbol{g}_{n,k}^l \big) \end{aligned} \quad (20)$$

where $\boldsymbol{g}_{n,k}^l = \boldsymbol{d}_k^l \otimes \sigma'(\mathbf{A}_n^l [1, \boldsymbol{c}_n^l]^T)$; "$\otimes$" denotes element-wise multiplication; $\tilde{\mathbf{A}}_n^l$ is the consequent parameter matrix $\mathbf{A}_n^l$ but with the first column removed; $\sigma'(x) = \sigma(x) \cdot (1 - \sigma(x))$ is the derivative of $\sigma(x)$ with respect to x; $\frac{\partial \lambda_{n,k}^l}{\partial \boldsymbol{x}_k^l}$ and $\mathbf{U}_{n,k}^l$ are obtained by Eqs. (21) and (22).

$$\frac{\partial \lambda_{n,k}^l}{\partial \boldsymbol{x}_k^l} = \lambda_{n,k}^l \cdot \big(\frac{2(\boldsymbol{p}_n^l - \boldsymbol{x}_k^l)}{(\tau_n^l)^2} - \sum_{i=1}^{N^l} (\lambda_{i,k}^l \cdot \frac{2(\boldsymbol{p}_i^l - \boldsymbol{x}_k^l)}{(\tau_i^l)^2}) \big) \quad (21)$$

$$\mathbf{U}_{n,k}^l = [\boldsymbol{v}_{n,1}^l \otimes \sigma'(\mathbf{V}_n^l \boldsymbol{x}_k^l), ..., \boldsymbol{v}_{n,M^l}^l \otimes \sigma'(\mathbf{V}_n^l \boldsymbol{x}_k^l)] \quad (22)$$

Here, $\boldsymbol{v}_{n,k}^l = [v_{n,1,k}^l, v_{n,2,k}^l, ..., v_{n,C^l,k}^l]^T$ denotes the k^{th} column of $\mathbf{V}_n^l$; $k = 1, 2..., M^l$.

Since the outputs of the l^{th} ECNFIS serve as the inputs of the $l+1^{th}$ ECNFIS (namely, $\boldsymbol{y}_k^l = \boldsymbol{x}_k^{l+1}$ $\forall l = 1, 2, ..., L-1$), Eqs. (18) and (19) can be simplified in a more compact form:

$$\frac{\partial \boldsymbol{e}_k}{\partial \mathbf{A}_n^l} = \lambda_{n,k}^l \cdot \boldsymbol{g}_{n,k}^l \cdot [1, \boldsymbol{c}_{n,k}^l] \quad (23)$$

$$\frac{\partial \boldsymbol{e}_k}{\partial \mathbf{V}_n^l} = \lambda_{n,k}^l \cdot (((\tilde{\mathbf{A}}_n^l)^T \cdot \boldsymbol{g}_{n,k}^l) \otimes \sigma'(\mathbf{V}_n^l \boldsymbol{x}_k^l)) \cdot (\boldsymbol{x}_k^l)^T \quad (24)$$

Detailed derivations for Eqs. (20), (23) and (24) are given in Supplementary Section A.

Based on the respective derivatives, $\mathbf{A}_n^l$ and $\mathbf{V}_n^l$ are updated using Eq. (25).

$$\mathbf{A}_n^l \leftarrow \mathbf{A}_n^l - \gamma_o \cdot \frac{\partial \boldsymbol{e}_k}{\partial \mathbf{A}_n^l}; \quad \mathbf{V}_n^l \leftarrow \mathbf{V}_n^l - \gamma_o \cdot \frac{\partial \boldsymbol{e}_k}{\partial \mathbf{V}_n^l} \quad (25)$$

where γ_o is the learning rate and there is $\gamma_o = 1$ because normalised firing strengths have been considered in $\frac{\partial \boldsymbol{e}_k}{\partial \mathbf{A}_n^l}$ and $\frac{\partial \boldsymbol{e}_k}{\partial \mathbf{V}_n^l}$ such that the consequent parameters of the IF-THEN rules will be adjusted proportionally according to the respective levels of activation in regards to the current input. However, experienced users may further specify a particular learning rate for each layer of MECFNN.

From Eqs. (23)-(25) one can see that if a particular rule is not activated by $\boldsymbol{x}_k^l$, namely, $\lambda_{n,k}^l = 0$, its consequent parameters will remain unchanged as both $\frac{\partial \boldsymbol{e}_k}{\partial \mathbf{A}_n^l}$ and $\frac{\partial \boldsymbol{e}_k}{\partial \mathbf{V}_n^l}$ equal to 0. Thanks to the activation control scheme (by Condition 2), these inactivated IF-THEN rules at the current learning cycle will be excluded from output generation and consequent parameter updating. This mechanism effectively prevents overfitting and improves the computational efficiency of individual ECNFISs as well as the overall stacking ensemble.

By using Eqs. (20), (23)-(25), consequent parts of the cascading ECNFISs are updated in a backward, layer-wise manner and after that, MECFNN starts a new learning cycle to process the next input ($k \leftarrow k + 1$).

The system identification process of the proposed MECFNN is summarised by Algorithm 1 in the form of pseudo code.

C. Computational Complexity Analysis

A brief analysis on the computational complexity of MECFNN is presented in this subsection.

MECFNN learns from data in a sample-wise manner. Given a particular input sample $\boldsymbol{x}_k$, the computational complexity for MECFNN to self-improve in response to $\boldsymbol{x}_k$ is $O(\sum_{l=3}^{L} \hat{N}_k^l C^l M^l W^l + \sum_{l=1}^{L} \hat{N}_k^l C^l (M^l + W^l))$. Together, the overall computational complexity of the system identification process of MECFNN given K input samples is $O(\sum_{k=1}^{K} (\sum_{l=3}^{L} \hat{N}_k^l C^l M^l W^l + \sum_{l=1}^{L} \hat{N}_k^l C^l (M^l + W^l))$. The detailed analysis can be found in Supplementary Section B.

In contrast, for a standard EFS that uses the RLS-based algorithm for consequent parameter learning, the computational complexity of updating N fuzzy rules within its rule base individually given K input samples is $O(KNM^3)$, due to the recursive update of covariance matrices [45].

It can be concluded from the analysis above that the adaptive dimensionality compression and activation control schemes combined with the use of error propagation effectively reduce the computational complexity of MECFNN, particularly when the dimensionality of data is high.

III. Experimental Investigation

In this section, numerical examples based on a variety of public numerical and image benchmark datasets are presented to demonstrate the merits of the proposed MECFNN on high-dimensional, complex classification problems. The algorithms were developed on Matlab2019a platform and the performance was evaluated on a desktop with dual core i7 processor 3.80 GHz×2 and 32.0 GB RAM. Unless specifically declared otherwise, the reported numerical results were obtained as the average of 10 Monte Carlo experiments to allow a certain degree of randomness and hence, a fair comparison. The source code of MECFNN is available at: https://github.com/Gu-X/Multilayer-Jointly-Evolving-and-Compressing-Fuzzy-Neural-Network.

A. Configuration

1) Data Description: A total of 15 high-dimensional problems are exploited in experimental investigation, which include five large-scale non-stationary datasets for network intrusion detection (NSLKDD [46], UNSWNB15 [47], CICIDS2017, CICIDS2018 [48] and HIKARI2021 [49]), three popular image classification datasets (MNIST [50], FMNIST [51] and

Algorithm 1 MECFNN identification process

$k \leftarrow 1$
while ($\boldsymbol{x}_k$ is available) **do**
  $\boldsymbol{x}_k^1 \leftarrow \boldsymbol{x}_k$
  if ($k = 1$) **then**
    for $l = 1$ to L **do**
    ### *Stage 0* ###
      $N^l \leftarrow 1$
      initialise $\boldsymbol{\eta}^l$ and X^l by (4)
      initialise $\boldsymbol{p}_{N^l}^l$, $\mathbf{V}_{N^l}^l$ and $\mathbf{A}_{N^l}^l$ by (5)
      initialise $\boldsymbol{R}_n^l$ in the form of (1)
      initialise $\boldsymbol{C}_{N^l}^l$ by (7)
      produce $\boldsymbol{y}_k^l$ by (8)
      if $l \neq L$ **then**
        $\boldsymbol{x}_k^{l+1} \leftarrow \boldsymbol{y}_k^l$
      else
        $\boldsymbol{y}_k \leftarrow \boldsymbol{y}_k^l$
  else
    ### *Stage 1* ###
    for $l = 1$ to L **do**
      update $\boldsymbol{\eta}^l$ and X^l by (9)
      for $n = 1$ to N^l **do**
        calculate $\mu_n(\boldsymbol{x}_k^l)$ by (10)
      if (Condition 1 is satisfied) **then**
        $N^l \leftarrow N^l + 1$
        initialise $\boldsymbol{p}_{N^l}^l$, $\mathbf{V}_{N^l}^l$ and $\mathbf{A}_{N^l}^l$ by (5)
        initialise $\boldsymbol{R}_n^l$ in the form of (1)
        initialise $\boldsymbol{C}_{N^l}^l$ by (7)
        calculate $\mu_{N^l}(\boldsymbol{x}_k^l)$ by (10)
      else
        $n^* = \arg\max_{n=1,2,\ldots,n^*}(\mu_n(\boldsymbol{x}_k^l))$
        update $\boldsymbol{C}_{n^*}^l$ by (12)
        update $\mu_{n^*}(\boldsymbol{x}_k^l)$ by (10)
      estimate $\hat{N}_k^l$ using (13)
      for $n = 1$ to N^l **do**
        calculate $\lambda_{n,k}^l$ by (15)
      produce $\boldsymbol{y}_k^l$ by (2)
      if $l \neq L$ **then**
        $\boldsymbol{x}_k^{l+1} \leftarrow \boldsymbol{y}_k^l$
      else
        $\boldsymbol{y}_k \leftarrow \boldsymbol{y}_k^l$
    ### *Stage 2* ###
  calculate $\frac{\partial \boldsymbol{e}_k}{\partial \boldsymbol{y}_k}$ by (17)
  $\boldsymbol{d}_k^L \leftarrow \frac{\partial \boldsymbol{e}_k}{\partial \boldsymbol{y}_k}$
  for $l = L$ to 1 **do**
    calculate $\frac{\partial \boldsymbol{e}_k}{\partial \mathbf{A}_n^l}$ by (23)
    calculate $\frac{\partial \boldsymbol{e}_k}{\partial \mathbf{V}_n^l}$ by (24)
    update $\mathbf{A}_n^l$ and $\mathbf{V}_n^l$ by (25)
    if $l \neq 1$ **then**
      calculate $\boldsymbol{d}_k^{l-1}$ by (20)
  $k \leftarrow k + 1$

CIFAR10 [52]), and seven remote sensing datasets for scene classification (SIRIWHU [53], UCMerced [54], RSSCN7 [55], AID [56], PatternNet [57], SAT4 [58] and SAT6 [58]). Key information of the 15 high-dimensional problems is summarised in Supplementary Section C. The pixel values of the images have been normalised to the range of [0,1] a priori.

In running the experiments, for NSLKDD and UNSWNB15 datasets, the categorical attributes have been converted into numerical attributes using one-hot encoding [59]. Redundant attributes, attributes without any variance and records with missing values are removed from CICIDS2017 and CICIDS2018 datasets [60]. For HIKARI2021 dataset, problem-specific attributes are removed in advance [48], [49]. Following the common practice, all the numerical attributes of the five intrusion detection datasets have been standardised a priori such that all the attributes are on the same scale [47].

For MNIST, FMNIST and CIFAR10 datasets, MobileNet (MBN) and MobileNetV2 (MBN_2) [61] are employed for feature extraction due to their relatively smaller model size, lower computational complexity and higher performance. Both models with weights pre-trained on ImageNet are used directly without fine-tuning. Activations from the last fully connected layer of the two pre-trained DNNs are used as the feature vectors of the images, namely, one 1024×1 dimensional feature vector per image by MBN and one 1280×1 dimensional feature vector per image by MBN_2. The extracted feature vectors are normalised prior to the experiments. In addition, images of MNIST and FMNIST datasets are also vectorised into 784×1 dimensional vectors and used for experimenting.

For SIRIWHU, UCMerced, RSSCN7, AID and PatternNet datasets, three DCNN models popular for remote sensing scene classification, including ResNet50 [62], DenseNet121 [63] and InceptionV3 [64], are used for feature extraction. To learn more meaningful, discriminative semantic features from remote sensing scenes, the three DNNs pre-trained on ImageNet are fine-tuned on the NWPU45 dataset following the same procedure described in [65]. Three 1024×1 dimensional representations are extracted by the fine-tuned models from each input image, and then aggregated into a single feature vector by averaging, thereby enhancing the descriptive abilities. This aggregated feature vectors of the images are directly used for carrying out the experiments. Due to the smaller image size and the additional near-infrared channel, DNNs pre-trained on ImageNet are unsuitable for feature extraction from SAT-4 and SAT-6 images. Hence, these images are converted to greyscale and vectorised into 784×1 dimensional vectors for experiments, similar to MNIST and FMNIST.

2) Parameter Setting for MECFNN: As aforementioned, MECFNN self-organises its multi-layered model structure and parameters from data based on their ensemble properties and mutual distances. The two distinctive features that differentiate MECFNN from existing works are: *i)* adaptive dimensionality compression to learn more compact, meaningful representations; and *ii)* adaptive activation control to reduce computational complexity and prevent overfitting. Hence, users are required to predetermine the number of layers/ensemble components, L, the output sizes of each layer (except for the final layer), W_1, W_2, ..., W_{L-1}, the compression ratio of each

layer, ϵ^1,ϵ^2,...,ϵ^L, the threshold, δ_o used by Condition 1 for novel data pattern detection and the ratio, ρ_o used by Condition 2 for rule activation control.

It has to be stressed that these externally controlled parameters are not problem- or user- specific, and can be determined without prior knowledge of the problem. As the main purpose of this paper is to introduce the proposed concept and general principles, for simplicity, the compression ratios of the individual layers are set to be uniformly the same, namely, $\epsilon^1 = \epsilon^2 =, ..., = \epsilon^L = \epsilon_o$, and the output sizes of individual ensemble components are set as $W^l = W_o\epsilon_o^{l-L}$ $(l = 1, 2, ..., L-1)$, considering the decreasing dimensionality of the input layer-by-layer. In running the experiments, unless specifically declared otherwise, MECFNN with a two-layer architecture ($L = 2$) is considered for performance demonstration. The values of the other four externally controlled parameters, namely, W_o, ϵ_o, ρ_o and δ_o are set as: $W_o = 3W$, $\epsilon_o = 0.5$, $\rho_o = 0.95$ and $\delta_o = e^{-3}$.

Compared with existing single-model EFSs and EFS-based ensemble models, MECFNN has a more complex model structure and a greater amount of trainable parameters in exchange for the ability to learn more compact, meaningful, discriminative representations from data. Hence, to ensure that MECFNN is trained sufficiently, it is trained on the same training sets with random shuffling for 100 epochs during the experiments. However, note that the aforementioned configurations only serves as an option feasible for users to consider and it is demonstrated through numerical examples that MECFNN with this configuration offers superior performances on a variety of high-dimensional, complex, challenging problems, surpassing the alternatives. Experienced users can further adjust the parameter settings accordingly utilising prior knowledge of the problem and domain expertise to maximise the performance of MECFNN.

To better understand the influences of the five externally controlled parameters (L, W_o, ϵ_o, ρ_o and δ_o) on the classification performance of the proposed MECFNN, a sensitivity analysis is carried out using the following seven datasets: NSLKDD, CICIDS2017, vectorised MNIST and FMNIST, SIRIWHU, RSSCN7 and PatternNet. The detailed results are presented in Supplementary Section D.

3) Methods for Performance Comparison: The following 12 popular single-model classification algorithms are involved in the numerical experiments for performance comparison on the aforementioned high-dimensional problems: 1) k-nearest neighbour (kNN) [66]; 2) decision tree (DT) [67]; 3) support vector machine (SVM) [68]; 4) multilayer perceptron (MLP); 5) recurrent neural network (RNN) [69]; 6) long short-term memory network (LSTM) [70]; 7) bidirectional LSTM network with attention mechanism (BAT) [59]; 8) parsimonious learning machine (PALM) [71]; 9) sequence classifier (SC) [72]; 10) leaky ReLU-based evolving classifier (LREC) [73]; 11) self-organising fuzzy belief inference system (SOFBIS) [74], and; 12) SEFIS [23]. Furthermore, the following 11 multi-model ensemble classifiers are also employed for performance comparison: 1) Adaboost.M2 DT-based ensemble classifier (ADBDT) [75]; 2) Adaboost.M2 kNN-based ensemble classifier (ADBNN) [75]; 3) random forest (RF) [76]; 4) stage-wise additive modeling using a multi-class exponential loss function (SAMME)-based DT ensemble classifier (SAMDT) [77]; 5) SAMME-based kNN ensemble classifier (SAMNN) [77]; 6) eEnsemble (eEns) [28]; 7) XGBoost [78]; 8) self-organising fuzzy inference ensemble system (SOFEns) [31]; 9) fuzzily weighted adaptive boosting (FWADB) [34]; 10) self-training hierarchical prototype-based ensemble framework (STHPEF) [65], and; 11) MEFNN [41]. Hence, a total of 23 SOTA single-model and multi-model classification algorithms are comparatively examined. Parameter settings of the comparative classifiers are given in Supplementary Section E. Note that STHPEF is designed for scene classification on regular-sized remote sensing images specifically and, hence, it will be only involved in numerical examples on SIRIWHU, UCMerced, RSSCN7, AID and PatternNet datasets.

B. Numerical Example on Network Intrusion Detection

First, the performance of MECFNN is evaluated on the five numerical datasets for network intrusion detection, which include NSLKDD, UNSWNB15, CICIDS2017, CICIDS2018 and HIKARI2021. Since the sizes of the five datasets are all very large, random down-sampling is performed to facilitate simulation whilst preserving the original distributions of data. In this study, for NSLKDD and UNSWNB15 datasets, the original training-testing splits are kept, and 10% of the training and testing data are randomly selected for running the experiment each time. For CICIDS2017, CICIDS2018 and HIKARI2021, 1%, 0.2% and 5% of data are randomly selected and divided into the training and testing sets with the split ratio of 1:1 in each experiment. The classification results obtained by MECFNN on the testing data of the five datasets after are reported in Table I in terms of accuracy (acc) and balanced accuracy ($bacc$). In addition, the performances of MECFNN with a three layer architecture, re-denoted as MECFNN^3 (all other externally controlled parameters follow the recommended setting) are also reported in the same table for better demonstration. The results obtained by 22 SOTA single-model and ensemble competitors under the same experimental settings are given in Table I for comparison, where the best results are in bold.

Table I shows that MECFNN and MECFNN^3 achieve high-level classification performances on the five network intrusion detection datasets, outperforming the majority of SOTA competitors including mainstream ANN models, e.g., RNN, LSTM, BAT, EFS models, e.g., PALM, LREC, SOFBIS, and recently introduced EFS-based ensemble classifiers, namely, FWADAB and MEFNN (its predecessor) in both acc and $bacc$. Furthermore, one may notice that by increasing the depth of the model, MECFNN attains greater multi-level representation learning capabilities and achieves better performance on UNSWNB15 and CICIDS2017. It has to be admitted that DT and DT-based ensemble models such as RF and XGB are among the most popular approaches for network intrusion detection [79] and, indeed, they achieve better classification performances in this example than other types of classifiers (due to the nature of data), this example demonstrates the efficacy of MECFNN on these high-dimensional non-stationary

TABLE I
PERFORMANCE COMPARISON ON FIVE NETWORK INTRUSION DETECTION PROBLEMS

Algorithm	NSLKDD		UNSWNB15		CICIDS2017		CICIDS2018		HIKARI2021	
	acc	$bacc$	acc	$bacc$	acc	$bacc$	acc	$bacc$	acc	$bacc$
MECFNN	0.7883	0.8065	0.8485	0.8349	0.9743	0.9557	0.9848	0.9638	**0.9319**	0.6024
MECFNN^3	0.7848	0.8039	0.8489	0.8352	0.9768	0.9844	0.9583	0.9633	0.9317	0.6024
kNN	0.7780	0.7965	0.8414	0.8283	0.9871	0.9814	0.9867	0.9683	0.9194	0.6436
DT	0.7946	**0.8125**	0.8615	0.8507	0.9938	0.9911	0.9831	0.9664	0.9138	0.6686
SVM	0.7577	0.7779	0.8111	0.7905	0.9299	0.8959	0.9424	0.8693	0.9307	0.5055
MLP	0.7881	0.8046	0.8448	0.8301	0.9789	0.9628	0.9854	0.9672	0.9316	0.6039
RNN	0.7925	0.8078	0.8197	0.8047	0.9242	0.8820	0.9556	0.9082	0.9245	0.5418
LSTM	0.7927	0.8103	0.8221	0.8108	0.9268	0.8746	0.9140	0.8793	0.9121	0.6042
BAT	0.7531	0.7739	0.8191	0.8007	0.9427	0.8805	0.9524	0.8961	0.9306	0.5484
PALM	0.7588	0.7709	0.7922	0.7716	0.9007	0.7621	0.9396	0.8578	0.9297	0.5110
SC	0.7866	0.8086	0.8664	**0.8576**	0.9808	0.9606	0.9615	0.9114	0.9169	0.6363
LREC	0.7638	0.7842	0.8091	0.7882	0.8992	0.7578	0.9366	0.8494	0.9302	0.5123
SOFBIS	0.7550	0.7800	0.8455	0.8326	0.9634	0.9318	0.9776	0.9450	0.9252	0.6302
SEFIS	0.7806	0.7937	0.6499	0.6356	0.7361	0.6092	0.7300	0.6154	0.8371	0.5944
ADBDT	0.7952	0.8123	0.8671	0.8546	0.9946	0.9913	0.9873	0.9687	0.9306	0.6204
ADBNN	0.7781	0.7965	0.8397	0.8263	0.9874	0.9814	0.9869	0.9680	0.9206	0.6401
RF	0.7744	0.7981	0.8347	0.8171	0.9942	0.9894	0.9857	0.9674	0.9314	0.6109
SAMDT	**0.7946**	**0.8125**	0.8657	0.8548	0.9838	0.9911	0.9840	0.9669	0.9229	0.6279
SAMNN	0.7780	0.7965	0.8414	0.8283	0.9871	0.9814	0.9867	0.9683	0.9194	0.6436
eEns	0.6666	0.7021	0.7207	0.7123	0.7074	0.7216	0.7100	0.6893	0.7373	**0.7393**
XGB	0.7878	0.8098	**0.8687**	0.8562	**0.9964**	**0.9945**	**0.9874**	**0.9695**	0.9245	0.6389
SOFEns	0.7783	0.7975	0.8364	0.8230	0.9621	0.9643	0.9725	0.9608	0.9243	0.5927
FAWADB	0.7818	0.8024	0.8358	0.8216	0.9655	0.9407	0.9852	0.9614	0.9305	0.5079
MEFNN	0.7756	0.7941	0.8416	0.8263	0.9722	0.9506	0.9830	0.9631	0.9317	0.5859

problems over ANNs, conventional EFS and EFS-based ensemble classifiers as well as other mainstream classifiers such as SVM, kNN, etc.

C. Numerical Example on Image Classification

Next, numerical experiments on the three popular image classification problems, namely, MNIST, FMNIST and CIFAR10, are carried out for performance evaluation. In running the experiments, MECFNN is firstly trained on the feature vectors of the training images extracted by pre-trained MBN and then tested on the feature vectors extracted from the testing images. To facilitate simulation, in each experiment, only the feature vectors of 10000 randomly selected training images are used for training the classification model. The classification results obtained by MECFNN in terms of acc on the three image datasets are reported in Table II. The three-layer MECFNN^3 and 22 single-model/ensemble classifiers involved in the previous example are also used for performance comparison under the same experimental protocol. In addition, the same experiments are repeated with the feature vectors of images extracted by pre-trained MBN_2, and the obtained results by the 24 classification algorithms are reported in the same table. The experimental results obtained on the vectorised images of MNIST and FMNIST datasets under the same protocol are also given in Table II for better comparison. The best results are in bold.

It can be observed from Table II that MECFNN achieves the highest classification performance on CIFAR10 with the feature vectors extracted by MBN_2, and it average acc over the three image datasets is 0.8114, which is the highest among the 24 classifiers involved in this example. By contrast, MEFNN, SVM and MECFNN^3 achieve the second, third and forth highest overall performances with the average acc of 0.8087, 0.8045 and 0.7973, respectively.

D. Numerical Example on Scene Classification

Then, the classification performance of MECFNN is evaluated on seven remote sensing scene classification problems, namely, SIRIWHU, RSSCN7, UCMerced, AID, PatternNet, SAT-4 and SAT-6. In running the experiments, for SIRIWHU dataset, the training-testing split is set to be 4:1. Two training-testing split ratios, namely, 1:4 and 1:1, are considered for RSSCN7 and AID datasets and, for UCMerced, the split ratios are set as 1:1 and 4:1, following the common practice [53], [56]. To facilitate simulation, in each experiment, 160 images per class are selected randomly from PatternNet dataset for training the classification models and another 40 images per class are selected for performance evaluation, following the commonly used splitting ratio of 4:1 [65]. For SAT-4 and SAT-6 datasets, 1000 images per class are selected from the training and testing sets, with 500 images from each, to carry out the numerical experiments, which is equivalent to setting the splitting ratio to 1:1. The scene classification performances of MECFNN and the aforementioned 23 comparative classification algorithms on the remote sensing datasets are reported in Table III in terms of acc. Similar, the best result per dataset is highlighted.

One can see from Table III that MECFNN offers the third best classification performance over the seven benchmark datasets with the overall average acc of 0.8821, slightly lower than XGB and RF (due to their significantly higher performances on SAT-4 and SAT-6 datasets). It is also suggested by Table III that MECFNN tends to overfit and performs worse on datasets with many classes but few images per class, e.g.,

TABLE II
PERFORMANCE COMPARISON ON THREE IMAGE CLASSIFICATION PROBLEMS

Algorithm	MNIST			FMNIST			CIFAR10	
	Orig	MBN	MBN_2	Orig	MBN	MBN_2	MBN	MBN_2
MECFNN	0.9570	0.9724	0.9715	0.8480	0.8610	0.8674	0.5069	**0.5070**
$MECFNN^3$	0.9443	0.9589	0.9646	0.8368	0.8457	0.8495	0.4946	0.4876
kNN	0.9487	0.9602	0.9560	0.8134	0.8347	0.8311	0.3615	0.3590
DT	0.8103	0.7808	0.7798	0.7521	0.6553	0.6491	0.2531	0.2583
SVM	0.9219	0.9772	0.9745	0.8241	**0.8718**	0.8626	0.5152	0.4884
MLP	**0.9617**	0.9664	0.9730	0.8474	0.8335	0.8591	0.4499	0.4662
RNN	0.9560	0.9520	**0.9751**	0.8353	0.8335	0.8680	0.2733	0.4694
LSTM	0.9633	0.9572	**0.9751**	0.8459	0.8393	0.8679	0.4024	0.4909
BAT	0.8963	0.9547	0.8677	0.7557	0.8200	0.7136	0.4589	0.4182
PALM	0.8454	**0.9784**	0.9735	0.8012	0.8716	0.8671	0.5108	0.4930
SC	0.9340	0.9531	0.9611	0.8341	0.8269	0.8331	0.3604	0.3589
LREC	0.8493	0.9776	0.9740	0.8027	0.8690	0.8677	**0.5201**	0.4998
SOFBIS	0.9466	0.9168	0.9069	0.8036	0.7745	0.7844	0.3984	0.3847
SEFIS	0.5194	0.5884	0.6912	0.5363	0.5127	0.4873	0.1721	0.1417
ADBDT	0.8943	0.8954	0.9014	0.8151	0.7644	0.7673	0.1257	0.1266
ADBNN	0.9488	0.9611	0.9566	0.8036	0.8278	0.8236	0.2994	0.3021
RF	0.9242	0.9253	0.9285	0.8263	0.8034	0.8025	0.4146	0.4165
SAMDT	0.8579	0.8778	0.8816	0.7917	0.7498	0.7540	0.3642	0.3669
SAMNN	0.9487	0.9602	0.9560	0.8134	0.8347	0.8311	0.3512	0.3477
eEns	0.7237	0.8217	0.8388	0.6695	0.7038	0.7074	0.2658	0.2723
XGB	0.9494	0.9545	0.9549	**0.8594**	0.8422	0.8400	0.4646	0.4674
SOFEns	0.9526	0.9632	0.9594	0.8208	0.8426	0.8417	0.4149	0.4123
FAWADB	0.9639	0.9648	0.9591	0.8447	0.8549	0.8469	0.4319	0.4320
MEFNN	0.9499	0.9733	0.9738	0.8486	0.8519	**0.8705**	0.5018	0.4996

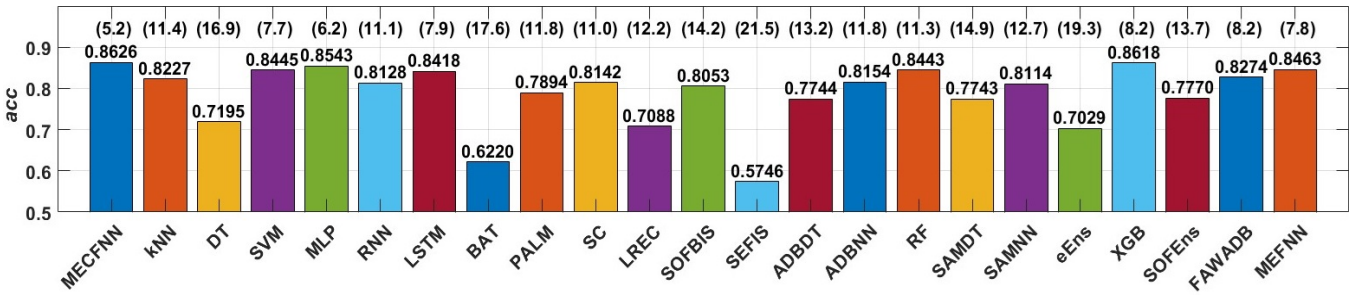


Fig. 3: Average acc rates and ranks of the 23 classification models over 15 high-dimensional classification problems.

UCMerced. Interestingly, one may note that LREC struggles to classify images when the number of classes exceeds 30.

E. Additional Numerical Results and Analysis

For better demonstration, the average classification performances of MECFNN and 22 SOTA single-model/ensemble classifiers in terms of acc on the 15 high-dimensional problems (23 sets of experiments in total) are shown in Fig.3 in the form of a bar chart. In addition, the performances of the 23 classification models are ranked per dataset (and per setting if different feature descriptors or split ratios are used for the same dataset) from the best (1^{st}) to the worst (23^{th}) based on acc. The average rank of each individual classification model is also indicated in Fig. 3, shown in brackets on top of the bar accordingly. One can see from Fig.3 that MECFNN offers the greatest classification performance over the 12 high-dimensional benchmark problems with an average acc of 0.8626 and rank of 5.2, outperforming all the single-model and ensemble competitors. This comparison demonstrates the superior performance of MECFNN on high-dimensional classification problems.

1) Statistical Significance Analysis. To examine whether the superior performance of MECFNN over the competitors is of statistical significance, pairwise Wilcoxon signed rank tests [80] are further carried out based on the acc rate per dataset (per experimental setting), and the p-values returned from the pairwise tests are reported in Supplementary Table S9. One can see from this table that 19 out of the 22 p-values reported are below the level of significance specified by $\alpha = 0.05$, suggesting that the performance of MECFNN is significantly better that others, consistent with the results presented in Fig. 3.

2) Diversity Demonstration. As aforementioned, the utilisation of SACP matrices effectively enhances the diversity. To show this, the absolute correlation coefficients between the vectorised SACP matrices associated with the IF-THEN rules learned by MECFNN as per the particular numerical experiment on the RSSCN7 dataset (with the splitting ratio of 1:4), in the form of matrix in Supplementary Figs. S1 and S2. These two figures illustrate that the maximum absolute correlation coefficients between any two SACP matrices are 0.02 for the first layer and 0.2 for the second. The low

TABLE III
PERFORMANCE COMPARISON ON FOUR REMOTE SENSING SCENE CLASSIFICATION PROBLEMS

Algorithm	SIRIWHU	RSSCN7		UCMerced		AID		PatternNet	SAT-4	SAT-6
	4:1	1:4	1:1	1:1	4:1	1:4	1:1	4:1	1:1	1:1
MECFNN	0.9454	0.9165	0.9316	0.9395	0.9612	0.9120	0.9336	0.9840	0.6921	0.6052
kNN	0.9202	0.8826	0.9104	0.9537	0.9683	0.8944	0.9211	0.9780	0.4354	0.4813
DT	0.7315	0.6200	0.6874	0.7375	0.7552	0.7397	0.7745	0.8079	0.6184	0.5899
SVM	**0.9477**	0.9160	0.9399	0.9582	0.9719	0.9173	0.9308	**0.9845**	0.5509	0.4993
MLP	0.9460	0.9124	0.9326	0.9409	0.9750	0.9037	0.9347	0.9835	0.5707	0.6633
RNN	0.9294	0.9163	0.9165	0.9566	0.9660	0.8824	0.8884	0.9499	0.2500	0.4601
LSTM	0.9498	**0.9215**	0.9277	0.9689	0.9740	0.9278	0.9140	0.9672	0.5325	0.5677
BAT	0.4689	0.5935	0.6551	0.2625	0.2844	0.2345	0.2495	0.1798	0.6108	0.4840
PALM	0.9385	0.8817	0.8681	0.4474	0.9781	**0.9261**	**0.9462**	0.7291	0.4362	0.3438
SC	0.9279	0.9049	0.9250	0.9651	0.9750	0.8968	0.9175	0.9786	0.3071	0.3548
LREC	0.9419	0.8945	0.9046	0.9530	**0.9793**	0.0360	0.0360	0.0263	0.4586	0.3739
SOFBIS	0.8481	0.8797	0.8864	0.9296	0.9367	0.8355	0.8433	0.9573	0.4699	0.5523
SEFIS	0.6021	0.5835	0.7326	0.7368	0.7536	0.4947	0.5561	0.7005	0.3486	0.3253
ADBDT	0.8427	0.8144	0.8686	0.8501	0.9026	0.7540	0.8311	0.6917	0.6983	0.6938
ADBNN	0.9196	0.8900	0.9170	0.9544	0.9686	0.8819	0.9114	0.9769	0.4203	0.4791
RF	0.9115	0.8775	0.8773	0.9515	0.9593	0.8897	0.9074	0.9578	0.7843	0.7405
SAMDT	0.7971	0.8018	0.8306	0.7608	0.8064	0.6769	0.7932	0.8480	0.6574	0.6429
SAMNN	0.9202	0.8887	0.9211	0.8456	0.9157	0.8826	0.9104	0.9113	0.4346	0.4764
eEns	0.7723	0.7887	0.8093	0.8610	0.8602	0.7709	0.7912	0.9026	0.5420	0.5227
XGB	0.9218	0.8879	0.9106	0.9347	0.9528	0.8789	0.9091	0.9649	**0.7953**	**0.7691**
SOFEns	0.8565	0.6097	0.7302	0.7976	0.7968	0.7361	0.7965	0.9098	0.4626	0.4931
FAWADB	0.9375	0.9164	0.9338	0.9628	0.9729	0.9070	0.9292	0.9827	0.3071	0.3844
MEFNN	0.9404	0.9023	0.9284	0.8421	0.8814	0.8683	0.9418	0.9061	0.6852	0.5950
STHPEF	0.9356	0.9176	**0.9358**	**0.9702**	0.9755	0.9139	0.9305	0.9824	-	-

correlation coefficients suggest a high degree of diversity between these SACP matrices, indicating that they project the data onto different lower dimensional sub-spaces, thereby enhancing the diversity within the stacking ensemble.

3) Ablation Analysis. An ablation analysis is further performed to demonstrate the influence of the adaptive dimensionality compression scheme and adaptive activation control scheme on the performance of MECFNN. The detailed results are presented in Supplementary Section G. The ablation analysis demonstrates the superior performance of the proposed MECFNN over the three alternatives and its predecessor, highlighting the effectiveness of the proposed two schemes. In particular, the adaptive dimensionality compression scheme enables the model to capture more discriminative features using fewer IF-THEN rules (although this is at the expense of increased computational complexity and potentially, a greater susceptibility to overfitting, if applied alone). However, the adaptive activation control scheme embedded in the model helps reduce overfitting while reinforcing classification accuracy, by dynamically filtering less activated IF-THEN rules from parameter updating and output generation. This filtering process also improves the computational efficiency of MECFNN. The combination of the two schemes effectively enhances the representation learning of MECFNN and promotes generalisation without a significant increase in computational complexity overall. The statistical analysis given by Supplementary Table S11 further indicates that the performance improvement is of statistical significance.

F. Discussions

To summarise, the systematic numerical experiments conducted in this study have collectively and consistently demonstrated the superior performance of the proposed MECFNN over the SOTA classifiers, including its predecessor MEFNN for high-dimensional data classification. Thanks to the novel adaptive activation control and adaptive dimensionality compression schemes, MECFNN learns highly compact, informative and meaningful multi-level representations from data with high dimensionality and complex structure, and attains high generalisation by dynamically dropping out less activated IF-THEN rules. As a result, MECFNN is able to achieve greater classification performance on a variety of popular high-dimensional complex classification problems of different natures, outperforming the 23 single-model/ensemble competitors. However, MECFNN has higher computational complexity when compared to its predecessor, due to the additional costs of dimensionality compression and compressive projection matrix updating. Similar to its predecessor, MECFNN also requires more training data to attain excellent performance than alternative EFS models that employ RLS-based algorithms for consequent parameter updating.

Note that as the initial report on a novel approach, the main purpose of this study is to introduce the relevant key concepts and general principles. Thus, all the numerical results of MECFNN reported in this section are obtained using the recommended parameter setting specified previously without any tuning, facilitating fair comparisons. However, the recommended parameter setting may be far from optimal, depending on the nature of data. In fact, it can be seen from Supplementary Tables S4 and S5 that reducing the value of W_o or increasing the value of ϵ_o can greatly improve the computational efficiency of MECFNN without substantial decrease in its classification accuracy. However, this is not the case when the dataset contains many classes, e.g., PatternNet. Supplementary Tables S6 and S7 demonstrate that by reducing

the value of ρ_o or δ_o, MECFNN performs better on MNIST, but worse on FMNIST. Although it is practically impossible to find an universal parameter setting that works well for all problems, the recommended parameter setting can serve as the starting point. Indeed, the performance of MECFNN can be maximised by adjusting the externally controlled parameters, including modifying the structure by adding or removing a certain number of layers. Hence, there is a large space for performance improvement by optimising the externally controlled parameters.

IV. Conclusion

This paper has presented a novel multilayer evolving neural fuzzy ensemble model with adaptive dimensionality compression and activation control for high-dimensional data classification. The resultant model, named MECFNN is capable of self-organising and self-updating the underlying multilayer system structure and parameters from data on a sample-by-sample basis, whilst performing dimensionality compression simultaneously through the utilisation of SACP matrices. These SACP matrices are continuously updated from data during the learning process to minimise the prediction errors. This enables MECFNN to better preserve the discriminative features in the data and to effectively prevent losing key information during dimensionality compression. Systematic experimental investigations have been carried out, showcasing that MECFNN is superior to the SOTA approaches on various challenging high-dimensional problems, by offering greater classification accuracy.

Open issues remain to be further investigated, however.

1) Computational efficiency. The utilisation of SACP matrices effectively improves the classification performance of MECFNN, but inevitably increases its computational complexity. This is because the additional computational costs incurred by dimensionality compression and matrix updating. Generally, learning from high-dimensional data is itself a computationally expensive task, but more advanced approaches for dimensionality reduction/feature selection that are computationally efficient may help to minimise such complexity. This is of particular significance when the model interpretability is required in working towards explainable classification systems, especially when such approaches (e.g., [81]) have proven to be theoretically rigorous and empirically successful.

2) Consequent parameter learning. Similar to its predecessor MEFNN, MECFNN requires larger amounts of training data (or being trained repeatedly with the same data) to achieve excellent performance. This is restricted by the use of the error backpropagation algorithm for consequent part updating to avoid ambiguities in defining the error functions for individual layers. To address this common limitation shared with many SOTA techniques, one important direction for future research is to design a more effective solution for training the consequent parts of the IF-THEN fuzzy rules.

3) Stability analysis. Although the numerical examples presented in this study have demonstrated the effectiveness and validity of the proposed MECFNN, there is a lack of theoretical analysis on its stability and convergence. In particular, the use of SACP matrices in EFS-based stacking ensemble models trained by backpropagation has not been explored previously and the impact of dimensionality compression upon the stability and convergence of MECFNN remains unknown. A continued examination and development alone this direction is very interesting.

4) Structure evolving. Reflecting on the structure of MECFNN being able to self-evolve from data horizontally (by adding new rules to different layers) once the number of layers is fixed, it would be extremely useful to devise a mechanism that helps MECFNN to also self-evolve vertically. This may be implemented by autonomously adding/removing layers during the learning process for better approximation.

5) Externally controlled parameters. Finally, concerning less experienced problem domains, it would be helpful if MECFNN would be able to self-adapt certain externally controlled parameters (e.g., δ_o and ϵ_o) with respect to the nature of the domain data after the initial values have been provided.

References

[1] S. Ayesha et al., "Overview and comparative study of dimensionality reduction techniques for high dimensional data," *Inf. Fusion*, vol. 59, pp. 44–58, 2020.

[2] S. Erfani et al., "High-dimensional and large-scale anomaly detection using a linear one-class SVM with deep learning," *Pattern Recognit.*, vol. 58, pp. 121–134, 2016.

[3] Z. Han et al., "A review of deep learning models for time series prediction," *IEEE Sens. J.*, vol. 21, no. 6, pp. 7833–7848, 2021.

[4] S. Dong et al., "A survey on deep learning and its applications," *Comput. Sci. Rev.*, vol. 40, p. 100379, 2021.

[5] C. Rudin, "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead," *Nat. Mach. Intell.*, vol. 1, no. 5, pp. 206–215, 2019.

[6] B. Kim et al., "Why are saliency maps noisy? cause of and solution to noisy saliency maps," in *International Conference on Computer Vision Workshop*, 2019, pp. 4149–4157.

[7] G. Montavon et al., "Layer-wise relevance propagation: an overview," in *Explainable AI*: *interpreting, explaining and visualizing deep learning*, 2019, pp. 193–209.

[8] X. Bai et al., "Explainable deep learning for efficient and robust pattern recognition: a survey of recent developments," *Pattern Recognit.,* vol. 120, p. 108102, 2021.

[9] N. Kasabov, *Evolving connectionist systems*: *the knowledge engineering approach*. Springer Science & Business Media, 2007.

[10] P. Angelov, *Autonomous learning systems*: *from data streams to knowledge in real time*. John Wiley & Sons, Ltd., 2012.

[11] X. Gu and Q. Shen, "A self-adaptive fuzzy learning system for streaming data prediction," *Inf. Sci. (Ny).*, vol. 579, pp. 623–647, 2021.

[12] I. Skrjanc et al., "Evolving fuzzy and neuro-fuzzy approaches in clustering, regression, identification, and classification: a survey," *Inf. Sci. (Ny).*, vol. 490, pp. 344–368, 2019.

[13] P. Angelov and D. Filev, "An approach to online identification of Takagi-Sugeno fuzzy models," *IEEE Trans. Syst. Man, Cybern. - Part B Cybern.*, vol. 34, no. 1, pp. 484–498, 2004.

[14] H. Rong et al., "Sequential adaptive fuzzy inference system (SAFIS) for nonlinear system identification and prediction," *Fuzzy Sets Syst.*, vol. 157, no. 9, pp. 1260–1275, 2006.

[15] D. Leite, P. Costa, and F. Gomide, "Evolving granular neural networks from fuzzy data streams," *Neural Networks*, vol. 38, pp. 1–16, 2013.

[16] M. Pratama et al., "PANFIS: a novel incremental learning machine," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 25, no. 1, pp. 55–68, 2014.

[17] M. Pratama, S. Anavatti, and E. Lughofer, "Genefis: toward an effective localist network," *IEEE Trans. Fuzzy Syst.*, vol. 22, no. 3, pp. 547–562, 2014.

[18] D. Dovzan, V. Logar, and I. Skrjanc, "Implementation of an evolving fuzzy model (eFuMo) in a monitoring system for a waste-water treatment process," *IEEE Trans. Fuzzy Syst.*, vol. 23, no. 5, pp. 1761–1776, 2015.

[19] D. Ge and X. Zeng, "A self-evolving fuzzy system which learns dynamic threshold parameter by itself," *IEEE Trans. Fuzzy Syst.*, vol. 27, no. 8, pp. 1625–1637, 2018.

[20] M. Ferdaus et al., "PALM: an incremental construction of hyperplanes for data stream regression," *IEEE Trans. Fuzzy Syst.*, vol. 27, no. 11, pp. 2115–2129, 2019.

[21] D. Ge and X. Zeng, "Learning data streams online - an evolving fuzzy system approach with self-learning/adaptive thresholds," *Inf. Sci. (Ny).*, vol. 507, pp. 172–184, 2020.

[22] H. Huang et al., "Jointly evolving and compressing fuzzy system for feature reduction and classification," *Inf. Sci. (Ny).*, vol. 579, pp. 218–230, 2021.

[23] Z. Yang et al., "Statistically evolving fuzzy inference system for non-Gaussian noises," *IEEE Trans. Fuzzy Syst.*, vol. 30, no. 4, pp. 2649–2664, 2022.

[24] J. Garibaldi, "The need for fuzzy AI," *IEEE/CAA J. Autom. Sin.*, vol. 6, no. 3, pp. 610–622, 2019.

[25] D. Wu et al., "On the functional equivalence of TSK fuzzy systems to neural networks, mixture of experts, CART, and stacking ensemble regression," *IEEE Trans. Fuzzy Syst.*, vol. 28, no. 10, pp. 2570–2580, 2020.

[26] X. Gong et al., "Embedded feature selection approach based on TSK fuzzy system with sparse rule base for high-dimensional classification problems," *Knowledge-Based Syst.*, vol. 295, p. 111809, 2024.

[27] S. Feng et al., "On the accuracy-complexity trade-off of fuzzy broad learning system," *IEEE Trans. Fuzzy Syst.*, vol. 29, no. 10, pp. 2963–2974, 2021.

[28] J. Iglesias, A. Ledezma, and A. Sanchis, "Ensemble method based on individual evolving classifiers," in *IEEE Conference on Evolving and Adaptive Intelligent Systems*, 2013, pp. 56–61.

[29] M. Pratama, W. Pedrycz, and E. Lughofer, "Evolving ensemble fuzzy classifier," *IEEE Trans. Fuzzy Syst.*, vol. 26, no. 5, pp. 2552–2567, 2018.

[30] M. Pratama et al., "Online tool condition monitoring based on parsimonious ensemble+," *IEEE Trans. Cybern.*, vol. 50, no. 2, pp. 664–677, 2020.

[31] X. Gu et al., "Self-organizing fuzzy inference ensemble system for big streaming data classification," *Knowledge-Based Syst.*, vol. 218, p. 106870, 2021.

[32] J. Jiang et al., "Dynamic incremental ensemble fuzzy classifier for data streams in green internet of things," *IEEE Trans. Green Commun. Netw.*, vol. 6, no. 3, pp. 1316–1329, 2022.

[33] X. Gu, "Self-adaptive fuzzy learning ensemble systems with dimensionality compression from data streams," *Inf. Sci. (Ny).*, vol. 634, pp. 382–399, 2023.

[34] X. Gu and P. Angelov, "Multi-class fuzzily weighted adaptive boosting-based self-organising fuzzy inference ensemble systems for classification," *IEEE Trans. Fuzzy Syst.*, vol. 30, no. 9, pp. 3722–3735, 2022.

[35] E. Lughofer and M. Pratama, "Online sequential ensembling of predictive fuzzy systems," *Evol. Syst.*, vol. 13, no. 2, pp. 361–386, 2022.

[36] Z. Zhou and J. Feng, "Deep forest," *Natl. Sci. Rev.*, vol. 6, no. 1, pp. 74–86, 2019.

[37] Y. Li, H. Zhang, X. Xue, Y. Jiang, and Q. Shen, "Deep learning for remote sensing image classification: a survey," *Wiley Interdiscip. Rev.: Data Min. Knowl. Discov.*, vol. 8, no. 6, e1264, 2018.

[38] M. Pratama, W. Pedrycz, and G. Webb, "An incremental construction of deep neuro fuzzy system for continual learning of nonstationary data streams," *IEEE Trans. Fuzzy Syst.*, vol. 28, no. 7, pp. 1315–1328, 2020.

[39] X. Gu, "Multilayer ensemble evolving fuzzy inference system," *IEEE Trans. Fuzzy Syst.*, vol. 29, no. 8, pp. 2425–2431, 2021.

[40] L. Hu et al., "Hierarchical evolving fuzzy system: a method for multidimensional chaotic time series online prediction," *IEEE Trans. Fuzzy Syst.*, DOI: 10.1109/TFUZZ.2023.3348847, 2024.

[41] X. Gu et al., "Multilayer evolving fuzzy neural networks," *IEEE Trans. Fuzzy Syst.*, vol. 31, no. 12, pp. 4158–4169, 2023.

[42] T. Zhao, H. Cao, and S. Dian, "A self-organized method for a hierarchical fuzzy logic system based on a fuzzy autoencoder," *IEEE Trans. Fuzzy Syst.*, vol. 30, no. 12, pp. 5104–5115, 2022.

[43] P. Li et al., "Very sparse random projections," in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2006, pp. 287–296.

[44] H. Huang et al., "Multilayer stacked evolving fuzzy system combined with compressed representation learning," *IEEE Trans. Fuzzy Syst.*, vol. 32, no. 4, pp. 2223–2234, 2024.

[45] S. Cook, "An overview of computational complexity," *Commun. ACM.*, vol. 26, no. 6, pp. 400–408, 1983.

[46] M. Tavallaee et al., "A detailed analysis of the KDD CUP 99 data set," in *IEEE Symposium on Computational Intelligence for Security and Defense Applications*, 2009, pp. 1–6.

[47] N. Moustafa and J. Slay, "The evaluation of network anomaly detection systems: statistical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data set," *Inf. Secur. J.*, vol. 25, no. 1–3, pp. 18–31, 2016.

[48] I. Sharafaldin et al., "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *International Conference on Information Systems Security and Privacy*, 2018, pp. 108–116.

[49] A. Ferriyan et al., "Generating network intrusion detection dataset based on real and encrypted synthetic attack traffic," *Appl. Sci.*, vol. 11, no. 17, p. 7868, 2021.

[50] L. Deng, "The MNIST database of handwritten digit images for machine learning research," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 141–142, 2012.

[51] H. Xiao et al., "Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms," *arXiv Prepr. arXiv1708.07747*, 2017.

[52] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

[53] B. Zhao et al., "Dirichlet-derived multiple topic scene classification model for high spatial resolution remote sensing imagery," *IEEE Trans. Geosci. Remote Sens.*, vol. 54, no. 4, pp. 2108–2123, 2016.

[54] Y. Yang and S. Newsam, "Bag-of-visual-words and spatial extensions for land-use classification," in *International Conference on Advances in Geographic Information Systems*, 2010, pp. 270–279.

[55] Q. Zou et al., "Deep learning based feature selection for remote sensing scene classification," *IEEE Geosci. Remote Sens. Lett.*, vol. 12, no. 11, pp. 2321–2325, 2015.

[56] G. Xia et al., "AID: a benchmark dataset for performance evaluation of aerial scene classification," *IEEE Trans. Geosci. Remote Sens.*, vol. 55, no. 7, pp. 3965–3981, 2017.

[57] W. Zhou et al., "PatternNet: a benchmark dataset for performance evaluation of remote sensing image retrieval," *ISPRS J. Photogramm. Remote Sens.*, vol. 145, pp. 197–209, 2018.

[58] S. Basu et al., "DeepSat - a learning framework for satellite imagery," in *ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2015, no. 37, pp. 1–10.

[59] T. Su et al., "BAT: deep learning methods on network intrusion detection using NSL-KDD dataset," *IEEE Access*, vol. 8, pp. 29575–29585, 2020.

[60] M. Verkerken et al., "Towards model generalization for intrusion detection: unsupervised machine learning techniques," *J. Netw. Syst. Manag.*, vol. 30, no. 1, pp. 1–25, 2022.

[61] M. Sandler et al., "MobileNetV2: inverted residuals and linear bottlenecks," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4510–4520.

[62] K. He et al., "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[63] G. Huang et al., "Densely connected convolutional networks," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 4700–4708.

[64] C. Szegedy et al., "Rethinking the inception architecture for computer vision," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2818–2826.

[65] X. Gu et al., "A self-training hierarchical prototype-based ensemble framework for remote sensing scene classification," *Inf. Fusion*, vol. 80, pp. 179–204, 2022.

[66] P. Cunningham and S. Delany, "K-nearest neighbour classifiers," *Mult. Classif. Syst.*, vol. 34, pp. 1–17, 2007.

[67] J. Quinlan, "Induction of decision trees," *Mach. Learn.*, vol. 1, no. 1, pp. 81–106, 1986.

[68] N. Cristianini and J. Shawe-Taylor, *An introduction to support vector machines and other kernel-based learning methods*. Cambridge: Cambridge University Press, 2000.

[69] C. Yin et al., "A deep learning approach for intrusion detection using recurrent neural betworks," *IEEE Access*, vol. 5, pp. 21954–21961, 2017.

[70] S. Hochreiter and J. Urgen Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[71] M. Ferdaus et al., "PALM: an incremental construction of hyperplanes for data stream regression," *IEEE Trans. Fuzzy Syst.*, vol. 27, no. 11, pp. 2115–2129, 2019.

[72] R. Patro et al., "Dictionary-based classifiers for exploiting feature sequence information and their application to hyperspectral remotely sensed data," *Int. J. Remote Sens.*, vol. 40, no. 13, pp. 4996–5024, 2019.

[73] M. Ferdaus et al., "Significance of activation functions in developing an online classifier for semiconductor defect detection," *Knowledge-Based Syst.*, vol. 248, p. 108818, 2022.
[74] X. Gu, P. Angelov, and Q. Shen, "Self-organizing fuzzy belief inference system for classification," *IEEE Trans. Fuzzy Syst.*, vol. 30, no. 12, pp. 5473–5483, 2022.
[75] Y. Freund, R. Schapire, and M. Hill, "Experiments with a new boosting algorithm," in *International Conference on Machine Learning*, 1996, pp. 148–156.
[76] L. Breiman, "Random forests," *Mach. Learn. Proc.*, vol. 45, no. 1, pp. 5–32, 2001.
[77] J. Zhu et al., "Multi-class AdaBoost," *Stat. Interface*, vol. 2, no. 3, pp. 349–360, 2009.
[78] T. Chen and C. Guestrin, "Xgboost: a scalable tree boosting system," in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 785–794.
[79] M. Douiba et al., "An improved anomaly detection model for IoT security using decision tree and gradient boosting," *J. Supercomput.*, vol. 79, no. 3, pp. 3392–3411, 2023.
[80] M. Friedman, "A comparison of alternative tests of significance for the problem of m rankings," *Ann. Math. Stat.*, vol. 11, no. 1, pp. 86–92, 1940.
[81] R. Jensen and Q. Shen, "Semantics-preserving dimensionality reduction: rough and fuzzy-rough-based approaches," *IEEE Trans Knowl Data Eng*, vol. 16, no. 12, pp. 1457-1471, 2004.

Multilayer Evolving Fuzzy Neural Networks with Self-Adaptive Dimensionality Compression for High Dimensional Data Classification

Supplementary Materials

A. Detailed Derivation of Consequent Part Updating

Based on the derivative of the prediction error $\boldsymbol{e}_k$ with regards to the output $\boldsymbol{y}_k^L$ of the L^{th} ECNFIS at the last layer of MECFNN, namely, $\boldsymbol{d}_k^L = \frac{\partial \boldsymbol{e}_k}{\partial \boldsymbol{y}_k^L} = (\boldsymbol{y}_k^L - \boldsymbol{r}_k)$, the derivative of prediction error with respect to the consequent parameter matrix $\mathbf{A}_n^L$ of the n^{th} IF-THEN rule, $\boldsymbol{R}_n^L$ is calculated by Eq. (S1) ($n = 1,2,\dots,N^L$).

$$\frac{\partial e_{k,i}}{\partial a_{n,i,j}^L} = d_{k,i}^L \cdot \frac{\partial y_k^L}{\partial a_{n,i,j}^L} = \lambda_{n,k}^L \cdot d_{k,i}^L \cdot \frac{\partial \sigma\left(\boldsymbol{a}_{n,i}^L [1, \boldsymbol{c}_{n,k}^L]^T\right)}{\partial a_{n,i,j}^L}$$
$$= \begin{cases} \lambda_{n,k}^L \cdot d_{k,i}^L \cdot \sigma'\left(\boldsymbol{a}_{n,i}^L [1, \boldsymbol{c}_{n,k}^L]^T\right), & if\ j = 0 \\ \lambda_{n,k}^L \cdot d_{k,i}^L \cdot \sigma'\left(\boldsymbol{a}_{n,i}^L [1, \boldsymbol{c}_{n,k}^L]^T\right) \cdot c_{n,k,j}^L, & else \end{cases} \tag{S1}$$

where $i = 1,2,\dots,W^L$; $j = 1,2,\dots,C^L$; $\boldsymbol{d}_k^L = \left[d_{k,1}^L, d_{k,2}^L, \dots, d_{k,W^L}^L\right]^T$; $d_{k,i}^L = \frac{\partial e_{k,i}}{\partial y_{k,i}^L} = y_{k,i}^L - r_{k,i}$, and; $\boldsymbol{a}_{n,i}^L$ is the i^{th} row of $\mathbf{A}_n^L$.

The derivative of prediction error with respect to the self-adaptive compressive projection (SACP) matrix $\mathbf{V}_n^L$ of $\boldsymbol{R}_n^L$ is formulated in a similar from to Eq. (S1) as follows ($n = 1,2,\dots,N^L$).

$$\frac{\partial e_{k,i}}{\partial v_{n,i,t}^L} = d_{k,i}^L \cdot \frac{\partial y_{k,i}^L}{\partial c_{n,k,j}^L} \cdot \frac{\partial c_{n,k,j}^L}{\partial v_{n,j,t}^L} = \lambda_{n,k}^L \cdot d_{k,i}^L \cdot \frac{\partial \sigma\left(\boldsymbol{a}_{n,i}^L [1, \boldsymbol{c}_{n,k}^L]^T\right)}{\partial c_{n,k,j}^L} \cdot \frac{\partial \sigma\left(\bar{\boldsymbol{v}}_{n,j}^L \boldsymbol{x}_k^L\right)}{\partial v_{n,j,t}^L}$$
$$= \lambda_{n,k}^L \cdot d_{k,i}^L \cdot \sigma'\left(\boldsymbol{a}_{n,i}^L [1, \boldsymbol{c}_{n,k}^L]^T\right) \cdot a_{n,i,j}^L \cdot \sigma'\left(\bar{\boldsymbol{v}}_{n,j}^L \boldsymbol{x}_k^L\right) \cdot x_t^L \tag{S2}$$

where $i = 1,2,\dots,W^L$; $j = 1,2,\dots,C^L$; $t = 1,2,\dots,M^L$, and; $\bar{\boldsymbol{v}}_{n,j}^L$ is the j^{th} row of $\mathbf{V}_n^L$.

Eqs. (S1) and (S2) can be converted into a more compact form given by Eqs. (S3) and (S4):

$$\frac{\partial \boldsymbol{e}_k}{\partial \mathbf{A}_n^L} = \lambda_{n,k}^L \cdot \left(\boldsymbol{d}_k^L \otimes \sigma'\left(\mathbf{A}_n^L [1, \boldsymbol{c}_{n,k}^L]^T\right)\right) \cdot [1, \boldsymbol{c}_{n,k}^L] \tag{S3}$$

$$\frac{\partial \boldsymbol{e}_k}{\partial \mathbf{V}_n^L} = \lambda_{n,k}^L \cdot \left(\left(\left(\widetilde{\mathbf{A}}_n^L\right)^T \cdot \left(\boldsymbol{d}_k^L \otimes \sigma'\left(\mathbf{A}_n^L [1, \boldsymbol{c}_{n,k}^L]^T\right)\right)\right) \otimes \sigma'(\mathbf{V}_n^L \boldsymbol{x}_k^L)\right) \cdot (\boldsymbol{x}_k^L)^T \tag{S4}$$

To obtain the derivatives of prediction error with respect to the consequent parameter and SACP matrices, namely, $\mathbf{A}_n^{L-1}$ and $\mathbf{V}_n^{L-1}$ for the ECNFIS at the $L-1^{th}$ layer, $\boldsymbol{d}_k^{L-1}$ needs to be derived from $\boldsymbol{d}_k^L$ firstly using the chain rule:

$$\boldsymbol{d}_k^{L-1} = \frac{\partial \boldsymbol{e}_k}{\partial \boldsymbol{y}_k^L} \cdot \frac{\partial \boldsymbol{y}_k^L}{\partial \boldsymbol{y}_k^{L-1}} = \boldsymbol{d}_k^L \cdot \frac{\partial \boldsymbol{y}_k^L}{\partial \boldsymbol{x}_k^L} \tag{S5}$$

where $\frac{\partial \boldsymbol{y}_k^L}{\partial \boldsymbol{x}_k^L}$ is given by Eq. (S6) ($i = 1,2,\dots,W^L$; $j = 1,2,\dots,C^L$; $t = 1,2,\dots,M^L$):

$$\frac{\partial \boldsymbol{y}_{k,i}^L}{\partial \boldsymbol{x}_{k,t}^L} = \sum_{n=1}^{N^L} \left(\frac{\partial \lambda_{n,k}^L}{\partial \boldsymbol{x}_{k,t}^L} \cdot \sigma^T\left(\boldsymbol{a}_{n,i}^L [1, \boldsymbol{c}_{n,k}^L]^T\right) + \lambda_{n,k}^L \cdot \frac{\partial \sigma\left(\boldsymbol{a}_{n,i}^L [1, \boldsymbol{c}_{n,k}^L]^T\right)}{\partial \boldsymbol{x}_{k,t}^L}\right) \tag{S6}$$

and there are:

$$\frac{\partial \lambda_{n,k}^L}{\partial \boldsymbol{x}_{k,t}^L} = \lambda_{n,k}^L \left(\frac{2(p_{n,t}^L - \boldsymbol{x}_{k,t}^L)}{(\tau_n^L)^2} - \sum_{i=1}^{N^L} \left(\lambda_{i,k}^L \cdot \frac{2(p_{i,t}^L - \boldsymbol{x}_{k,t}^L)}{(\tau_i^L)^2}\right)\right) \tag{S7}$$

$$\frac{\partial \sigma\left(\boldsymbol{a}_{n,i}^L [1, \boldsymbol{c}_{n,k}^L]^T\right)}{\partial x_{k,t}^L} = \frac{\partial \sigma\left(\boldsymbol{a}_{n,i}^L [1, \boldsymbol{c}_{n,k}^L]^T\right)}{\partial \boldsymbol{c}_{n,k}^L} \cdot \frac{\partial \boldsymbol{c}_{n,k}^L}{\partial \boldsymbol{x}_{k,t}^L} = \sigma'\left(\boldsymbol{a}_{n,i}^L [1, \boldsymbol{c}_{n,k}^L]^T\right) \cdot \widetilde{\boldsymbol{a}}_{n,i}^L \cdot \left(\sigma'(\mathbf{V}_n^L \boldsymbol{x}_k^L) \otimes \boldsymbol{v}_{n,t}^L\right) \tag{S8}$$

Note that, $\boldsymbol{v}_{n,t}^{L}$ is the t^{th} column of $\mathbf{V}_n^L$.

Eq. (S6) is reformulated in a more compact form as:

$$\frac{\partial \boldsymbol{y}_k^L}{\partial \boldsymbol{x}_k^L} = \sum_{n=1}^{N^L}\left(\frac{\partial \lambda_{n,k}^L}{\partial \boldsymbol{x}_k^L}\cdot \sigma^T\left(\mathbf{A}_n^L\left[1,\boldsymbol{c}_{n,k}^L\right]^T\right) + \lambda_{n,k}^L \cdot \frac{\partial \sigma\left(\mathbf{A}_n^L\left[1,\boldsymbol{c}_{n,k}^L\right]^T\right)}{\partial \boldsymbol{x}_k^L}\right) \tag{S9}$$

where

$$\frac{\partial \lambda_{n,k}^L}{\partial \boldsymbol{x}_k^L} = \lambda_{n,k}^L\left(\frac{2\left(\boldsymbol{p}_n^L-\boldsymbol{x}^L\right)}{\left(\tau_n^L\right)^2} - \sum_{i=1}^{N^L}\left(\lambda_i^L\cdot\frac{2\left(\boldsymbol{p}_i^L-\boldsymbol{x}^L\right)}{\left(\tau_i^L\right)^2}\right)\right) \tag{S10}$$

$$\frac{\partial \sigma\left(\mathbf{A}_n^L\left[1,\boldsymbol{c}_{n,k}^L\right]^T\right)}{\partial \boldsymbol{x}_k^L} = \left(\widetilde{\mathbf{A}}_n^L\mathbf{U}_{n,k}^L\right)^T\cdot\sigma'\left(\mathbf{A}_n^L\left[1,\boldsymbol{c}_{n,k}^L\right]^T\right) \tag{S11}$$

$$\mathbf{U}_{n,k}^L = \left[\boldsymbol{v}_{n,1}^L\otimes\sigma'(\mathbf{V}_n^L\boldsymbol{x}_k^L), \boldsymbol{v}_{n,2}^L\otimes\sigma'(\mathbf{V}_n^L\boldsymbol{x}_k^L), \dots, \boldsymbol{v}_{n,M^L}^L\otimes\sigma'(\mathbf{V}_n^L\boldsymbol{x}_k^L)\right] \tag{S12}$$

Based on Eq. (S9), Eq. (S5) can be reformulated as:

$$\boldsymbol{d}_k^{L-1} = \sum_{n=1}^{N^L}\left(\frac{\partial \lambda_{n,k}^L}{\partial \boldsymbol{x}_k^L}\cdot\sigma^T\left(\mathbf{A}_n^L\left[1,\boldsymbol{c}_{n,k}^L\right]^T\right)\cdot\boldsymbol{d}_k^L + \left(\widetilde{\mathbf{A}}_n^L\mathbf{U}_{n,k}^L\right)^T\cdot\left(\boldsymbol{d}_k^L\otimes\sigma'\left(\mathbf{A}_n^L\left[1,\boldsymbol{c}_{n,k}^L\right]^T\right)\right)\right) \tag{S13}$$

and the derivatives of prediction error with respect to $\mathbf{A}_n^{L-1}$ and $\mathbf{V}_n^{L-1}$ are obtained as follows ($n = 1,2,\dots,N^{L-1}$).

$$\frac{\partial e_k}{\partial \mathbf{A}_n^{L-1}} = \lambda_{n,k}^{L-1}\cdot\left(\boldsymbol{d}_k^{L-1}\otimes\sigma'\left(\mathbf{A}_n^{L-1}\left[1,\boldsymbol{c}_{n,k}^{L-1}\right]^T\right)\right)\cdot\left[1,\boldsymbol{c}_{n,k}^{L-1}\right] \tag{S14}$$

$$\frac{\partial e_k}{\partial \mathbf{V}_n^{L-1}} = \lambda_{n,k}^{L-1}\cdot\left(\left(\left(\widetilde{\mathbf{A}}_n^{L-1}\right)^T\cdot\left(\boldsymbol{d}_k^{L-1}\otimes\sigma'\left(\mathbf{A}_n^{L-1}\left[1,\boldsymbol{c}_{n,k}^{L-1}\right]^T\right)\right)\right)\otimes\sigma'(\mathbf{V}_n^{L-1}\boldsymbol{x}_k^{L-1})\right)\cdot(\boldsymbol{x}_k^{L-1})^T \tag{S15}$$

Similarly, the derivatives of prediction errors with respect to $\mathbf{A}_n^l$ and $\mathbf{V}_n^l$ ($n = 1,2,\dots,N^l$) of the ECNFIS at the l^{th} layer of MECFNN can be calculated by Eqs. (S16)-(S18) ($\forall l = 1,2,\dots,L-2$):

$$\boldsymbol{d}_k^l = \sum_{n=1}^{N^{l+1}}\left(\frac{\partial \lambda_{n,k}^{l+1}}{\partial \boldsymbol{x}_k^{l+1}}\cdot\sigma^T\left(\mathbf{A}_n^{l+1}\left[1,\boldsymbol{c}_{n,k}^{l+1}\right]^T\right)\cdot\boldsymbol{d}_k^{l+1} + \left(\widetilde{\mathbf{A}}_n^{l+1}\mathbf{U}_{n,k}^{l+1}\right)^T\cdot\left(\boldsymbol{d}_k^{l+1}\otimes\sigma'\left(\mathbf{A}_n^{l+1}\left[1,\boldsymbol{c}_{n,k}^{l+1}\right]^T\right)\right)\right) \tag{S16}$$

$$\frac{\partial e_k}{\partial \mathbf{A}_n^l} = \lambda_{n,k}^l\cdot\left(\boldsymbol{d}_k^l\otimes\sigma'\left(\mathbf{A}_n^l\left[1,\boldsymbol{c}_{n,k}^l\right]^T\right)\right)\cdot\left[1,\boldsymbol{c}_{n,k}^l\right] \tag{S17}$$

$$\frac{\partial e_k}{\partial \mathbf{V}_n^l} = \lambda_{n,k}^l\cdot\left(\left(\left(\widetilde{\mathbf{A}}_n^l\right)^T\cdot\left(\boldsymbol{d}_k^l\otimes\sigma'\left(\mathbf{A}_n^l\left[1,\boldsymbol{c}_{n,k}^l\right]^T\right)\right)\right)\otimes\sigma'\left(\mathbf{V}_n^l\boldsymbol{x}_k^l\right)\right)\cdot\left(\boldsymbol{x}_k^l\right)^T \tag{S18}$$

B. Computational Complexity Analysis for MECFNN

Since MECFNN learns from data in a sample-wise manner, the computational complexity analysis is assumed to be conducted at the k^{th} time instance at which $\boldsymbol{x}_k$ is presented to MECFNN.

Stage 0 is for system initialisation and will not repeat after the first input sample $\boldsymbol{x}_1$ has been processed. Hence, the computational complexity of this stage is negligible.

Stage 1 is for structure evolving and antecedent part updating on a layer-by-layer basis. For the l^{th} ECNFIS ($l = 1,2,\dots,L$), the computational complexity of updating $\boldsymbol{\eta}^l$ and X^l given $\boldsymbol{x}_k^l$ is $O(M^l)$, and that of calculating the local density of $\boldsymbol{x}_k^l$ at the N^l clusters is $O(C^l M^l)$. The complexity of adding a new IF-THEN rule and the associated cluster to the system is $O(C^l(M^l+W^l))$, and that of updating the parameters of a cluster is $O(M^l)$. The computational complexity of output generation is $O\left(\widehat{N}_k^l C^l(M^l+W^l)\right)$. Therefore, the computational complexity of the l^{th} ECNFIS at Stage 1 is $O\left(\widehat{N}_k^l C^l(M^l+W^l)\right)$ and the overall computational complexity of MECFNN is $O\left(\sum_{l=1}^{L}\widehat{N}_k^l C^l(M^l+W^l)\right)$.

Stage 2 is for consequent part updating in response to $\boldsymbol{x}_k^l$, which include both the consequent parameter matrices and SACP matrices of individual ECNFISs within MECFNN. The complexity of calculating $\frac{\partial e_k}{\partial \mathbf{A}_n^l}$ and $\frac{\partial e_k}{\partial \mathbf{V}_n^l}$ is $O(C^l(M^l+W^l))$. The complexity of updating the consequent parts of the activated IF-THEN rules of the l^{th}

ECNFIS is $O\left(\widehat{N}_k^l C^l (M^l + W^l)\right)$, and that of calculating $\boldsymbol{d}_k^{l-1}$ from $\boldsymbol{d}_k^l$ is $O\left(\widehat{N}_k^l C^l M^l W^l\right)$. Note that there is $\boldsymbol{d}_k^L = \frac{\partial e_k}{\partial y_n^L}$. There also will be no need for calculating $\boldsymbol{d}_k^1$ since it will not be involved in parameter updating. Hence, the overall computational complexity of Stage 2 for MECFNN is $O\left(\sum_{l=3}^{L} \widehat{N}_k^l C^l M^l W^l + \sum_{l=1}^{L} \widehat{N}_k^l C^l (M^l + W^l)\right)$. Together, the overall computational complexity of the system identification process of MECFNN given K input samples is $O\left(\sum_{k=1}^{K}\left(\sum_{l=3}^{L} \widehat{N}_k^l C^l M^l W^l + \sum_{l=1}^{L} \widehat{N}_k^l C^l (M^l + W^l)\right)\right)$.

C. Key Information of Benchmark Datasets for Experimental Investigation

Table S1. Key information of five large-scale benchmark intrusion detection datasets

Dataset		#(Samples)	#(Attributes)	#(Normal Samples)	#(Anomalies)
NSLKDD	Training	125,973	38 numerical inputs + 3	67,343	58,630
	Testing	22,544	categorical inputs + 1 label	9711	12,833
UNSWNB15	Training	175,341	40 numerical inputs + 3	56,000	119,341
	Testing	82,332	categorical inputs + 1 label	37,000	45,332
CICIDS2017		2,830,743	78 numerical inputs + 1 label	2,273,097	557,646
CICIDS2018		16,232,944	79 numerical inputs + 1 label	13,484,708	2,748,236
HIKARI2021		555,278	83 numerical inputs + 1 label	517,582	37,696

Table S2. Key information of seven benchmark image datasets for classification

Dataset		#(Images)	#(Pixels)	#(Classes)
MNIST	Training	60,000	28×28×1	10
	Testing	10,000		
FMNIST	Training	60,000	28×28×1	10
	Testing	10,000		
CIFAR10	Training	50,000	32×32×3	10
	Testing	10,000		
SIRIWHU		2400	200×200×3	12
RSS		2800	400×400×3	7
UCM		2100	256×256×3	21
AID		10,000	600×600×3	30
PatternNet		30,400	256×256×3	38
SAT4	Training	400,000	28×28×4	4
	Testing	100,000		
SAT6	Training	324,000	28×28×4	6
	Testing	81,000		

D. Sensitivity Analysis

In this section, influences of the five externally controlled parameters, namely, L, W_o, ϵ_o, ρ_o and δ_o on the prediction performance of MECFNN are investigated using the following seven datasets: NSLKDD, CICIDS2017, vectorised MNIST and FMNIST, SIRIWHU, RSSCN7 and PatternNet. In running the experiments, the same protocols used in the numerical examples presented in Tables 1-3 are follows.

Firstly, the influence of L on the prediction performance of MECFNN is studied. In this example, W_o, ϵ_o, ρ_o and δ_o are fixed as $W_o = 3W$, $\epsilon_o = 0.5$, $\rho_o = 0.95$ and $\delta_o = e^{-3}$, following the recommended setting. The value of L is varied from 1 to 3. It is worth noting that MECFNN is reduced to a single-layer ECNFIS model if $L = 1$, and the value of W_o is set to be W in such case by default. The results obtained by MECFNN models with different L are tabulated in Table S3 in terms of acc, and number of rules per layer (N) and training time consumption per iteration (t_{tra}).

It is shown in Table S3 that, a deeper MECFNN will typically have more IF-THEN rules within the system and, therefore, a greater amount of antecedent and consequent parameters to be learned from data. In general, MECFNN with a deeper structure, namely, a larger L enables it to attain greater representation learning capabilities and achieve greater classification performance in terms of acc. On the other hand, a deeper MECFNN will need more training data to maximise its performance and the training process takes more time to be completed.

Hence, it can be observed from Table S3 that the two-layer MECFNN ($L = 2$) and three-layer MECFNN ($L = 3$) outperform the single-layer MECFNN ($L = 1$) in all experiments to a large degree on the two network intrusion detection datasets and the three remote sensing datasets. One may also notice that the two-layer MECFNN ($L = 2$) outperforms the three-layer MECFNN ($L = 3$) in all the experiments as well, but the difference between the performances of the two models in terms of acc gets smaller when more training data is presented.

Table S3. Classification performance of MECFNN with different settings of L

L	Meas.	NSLKDD	CICIDS2017	MNIST	FMNIST	SIRIWHU	RSSCN		PatternNet
							1:4	1:1	
1	acc	0.7559	0.9676	0.9580	0.8404	0.4946	0.4961	0.4992	0.5980
	(N^1)	(103.4)	(92.8)	(10.4)	(11.3)	(27.8)	(19.6)	(28.0)	(40.5)
	t_{tra}	49.0	24.4	479.0	365.3	173.1	32.1	130.0	847.2
2	acc	0.7883	0.9743	0.9570	0.8480	0.9454	0.9165	0.9316	0.9840
	$\begin{pmatrix} N^1 \\ N^2 \end{pmatrix}$	$\begin{pmatrix} 103.5 \\ 20.3 \end{pmatrix}$	$\begin{pmatrix} 92.8 \\ 29.2 \end{pmatrix}$	$\begin{pmatrix} 10.4 \\ 30.9 \end{pmatrix}$	$\begin{pmatrix} 11.3 \\ 20.0 \end{pmatrix}$	$\begin{pmatrix} 27.8 \\ 21.2 \end{pmatrix}$	$\begin{pmatrix} 19.7 \\ 18.5 \end{pmatrix}$	$\begin{pmatrix} 28.0 \\ 19.6 \end{pmatrix}$	$\begin{pmatrix} 40.5 \\ 39.7 \end{pmatrix}$
	t_{tra}	65.2	33.8	441.9	438.2	216.4	38.4	133.7	1088.1
3	acc	0.7805	0.9742	0.9428	0.8341	0.9096	0.8779	0.9159	0.9340
	$\begin{pmatrix} N^1 \\ N^2 \\ N^3 \end{pmatrix}$	$\begin{pmatrix} 103.4 \\ 24.4 \\ 16.0 \end{pmatrix}$	$\begin{pmatrix} 92.5 \\ 32.7 \\ 18.2 \end{pmatrix}$	$\begin{pmatrix} 10.4 \\ 35.2 \\ 33.8 \end{pmatrix}$	$\begin{pmatrix} 11.3 \\ 22.2 \\ 22.7 \end{pmatrix}$	$\begin{pmatrix} 27.8 \\ 23.0 \\ 27.5 \end{pmatrix}$	$\begin{pmatrix} 19.6 \\ 17.8 \\ 18.5 \end{pmatrix}$	$\begin{pmatrix} 28.0 \\ 20.0 \\ 16.5 \end{pmatrix}$	$\begin{pmatrix} 40.5 \\ 50.1 \\ 47.3 \end{pmatrix}$
	t_{tra}	85.4	47.5	423.7	405.5	199.5	47.7	127.5	1452.5

Secondly, the influence of W_o on the prediction performance of MECFNN is studied. In this example, L, ϵ_o, ρ_o and δ_o are fixed as L =2, $\epsilon_o = 0.5$, $\rho_o = 0.95$ and $\delta_o = e^{-3}$, following the recommended setting. The value of W_o is varied from W, $2W$, $3W$ and $4W$. The results obtained by MECFNN models with different parameter settings are tabulated in Table S4 in terms of acc, and number of rules per layer (N) and training time consumption per iteration (t_{tra}).

Table S4. Classification performance of MECFNN with different settings of W_o

W_o	Meas.	NSLKDD	CICIDS2017	MNIST	FMNIST	SIRIWHU	RSSCN		PatternNet
							1:4	1:1	
W	acc	0.7822	0.9735	0.9589	0.8450	0.9467	0.9166	0.9333	0.9419
	$\begin{pmatrix} N^1 \\ N^2 \end{pmatrix}$	$\begin{pmatrix} 103.4 \\ 15.0 \end{pmatrix}$	$\begin{pmatrix} 92.7 \\ 20.5 \end{pmatrix}$	$\begin{pmatrix} 10.4 \\ 39.5 \end{pmatrix}$	$\begin{pmatrix} 11.4 \\ 30.8 \end{pmatrix}$	$\begin{pmatrix} 27.7 \\ 38.5 \end{pmatrix}$	$\begin{pmatrix} 19.5 \\ 33.1 \end{pmatrix}$	$\begin{pmatrix} 28.0 \\ 31.3 \end{pmatrix}$	$\begin{pmatrix} 40.5 \\ 43.8 \end{pmatrix}$
	t_{tra}	28.4	11.4	321.4	209.4	94.5	25.5	84	577.7
$2W$	acc	0.7793	0.9756	0.9566	0.8487	0.9319	0.9203	0.9264	0.9729
	$\begin{pmatrix} N^1 \\ N^2 \end{pmatrix}$	$\begin{pmatrix} 103.4 \\ 21.2 \end{pmatrix}$	$\begin{pmatrix} 92.9 \\ 31.2 \end{pmatrix}$	$\begin{pmatrix} 10.5 \\ 35.7 \end{pmatrix}$	$\begin{pmatrix} 11.3 \\ 32.0 \end{pmatrix}$	$\begin{pmatrix} 27.7 \\ 25.4 \end{pmatrix}$	$\begin{pmatrix} 19.6 \\ 25.3 \end{pmatrix}$	$\begin{pmatrix} 28.0 \\ 21.7 \end{pmatrix}$	$\begin{pmatrix} 40.5 \\ 43.8 \end{pmatrix}$
	t_{tra}	43.4	23.4	403.0	331.2	113.3	33.5	120.3	730.1
$3W$	acc	0.7883	0.9743	0.9570	0.8480	0.9454	0.9165	0.9316	0.9840
	$\begin{pmatrix} N^1 \\ N^2 \end{pmatrix}$	$\begin{pmatrix} 103.5 \\ 20.3 \end{pmatrix}$	$\begin{pmatrix} 92.8 \\ 29.2 \end{pmatrix}$	$\begin{pmatrix} 10.4 \\ 30.9 \end{pmatrix}$	$\begin{pmatrix} 11.3 \\ 20.0 \end{pmatrix}$	$\begin{pmatrix} 27.8 \\ 21.2 \end{pmatrix}$	$\begin{pmatrix} 19.7 \\ 18.5 \end{pmatrix}$	$\begin{pmatrix} 28.0 \\ 19.6 \end{pmatrix}$	$\begin{pmatrix} 40.5 \\ 39.7 \end{pmatrix}$
	t_{tra}	65.2	33.8	441.9	438.2	216.4	38.4	133.7	1088.1
$4W$	acc	0.7873	0.9762	0.9549	0.8502	0.9421	0.9170	0.9277	0.9847
	$\begin{pmatrix} N^1 \\ N^2 \end{pmatrix}$	$\begin{pmatrix} 103.4 \\ 15.7 \end{pmatrix}$	$\begin{pmatrix} 92.8 \\ 33.7 \end{pmatrix}$	$\begin{pmatrix} 10.4 \\ 28.7 \end{pmatrix}$	$\begin{pmatrix} 11.3 \\ 18.8 \end{pmatrix}$	$\begin{pmatrix} 27.8 \\ 20.3 \end{pmatrix}$	$\begin{pmatrix} 19.6 \\ 17.8 \end{pmatrix}$	$\begin{pmatrix} 28.0 \\ 17.8 \end{pmatrix}$	$\begin{pmatrix} 40.5 \\ 42.2 \end{pmatrix}$
	t_{tra}	65.4	34.6	473.1	516.2	217.4	34.3	185	1220.6

Table S4 shows that W_o can significantly influence the performances of MECFNN by controlling the amount of information flowing between the layers. A larger W_o can facilitate the information exchange between layers. However, by increasing W_o, the number of parameters in the consequent parts of the IF-THEN rules of MECFNN grows, and the size of the antecedent parts of the rules (except for the ones at the first layer) increases as well. As a result, the computational complexity of MECFNN increases with a larger W_o, and the amount of training data needed for MECFNN to maximise its performance also increases. Hence, it can be observed from Table S4 that

the classification performance of MECFNN improves by increasing the values of W_o, particularly, on more complex problems with many classes, e.g., PatternNet, but the training time consumption increases as well.

Next, the influence of ϵ_o on the performance of MECFNN is investigated. In this example, L, W_o, ρ_o and δ_o are set as $L = 2$, $W_o = 3W$, $\rho_o = 0.95$ and $\delta_o = e^{-3}$, following the recommended setting. The value of ϵ_o is varied from 0.4, 0.5, 0.6 and 0.7. The results obtained by MECFNN models with different ϵ_o are reported in Table S5 in terms of acc, N and t_{tra}.

Table S5. Classification performance of MECFNN with different settings of ϵ_o

ϵ_o	Meas.	NSLKDD	CICIDS2017	MNIST	FMNIST	SIRIWHU	RSSCN		PatternNet
							1:4	1:1	
0.4	acc	0.7876	0.9756	0.9570	0.8483	0.9444	0.9145	0.9237	0.9785
	$\begin{pmatrix}N^1\\N^2\end{pmatrix}$	$\begin{pmatrix}103.4\\16.6\end{pmatrix}$	$\begin{pmatrix}92.5\\30.6\end{pmatrix}$	$\begin{pmatrix}10.5\\28.8\end{pmatrix}$	$\begin{pmatrix}11.3\\20.0\end{pmatrix}$	$\begin{pmatrix}27.6\\23.8\end{pmatrix}$	$\begin{pmatrix}19.6\\20.6\end{pmatrix}$	$\begin{pmatrix}27.9\\19.3\end{pmatrix}$	$\begin{pmatrix}40.5\\39.9\end{pmatrix}$
	t_{tra}	46.9	21.7	443.4	419.2	184.6	48.8	111.8	1025.0
0.5	acc	0.7883	0.9743	0.9570	0.8480	0.9454	0.9165	0.9316	0.9840
	$\begin{pmatrix}N^1\\N^2\end{pmatrix}$	$\begin{pmatrix}103.5\\20.3\end{pmatrix}$	$\begin{pmatrix}92.8\\29.2\end{pmatrix}$	$\begin{pmatrix}10.4\\30.9\end{pmatrix}$	$\begin{pmatrix}11.3\\20.0\end{pmatrix}$	$\begin{pmatrix}27.8\\21.2\end{pmatrix}$	$\begin{pmatrix}19.7\\18.5\end{pmatrix}$	$\begin{pmatrix}28.0\\19.6\end{pmatrix}$	$\begin{pmatrix}40.5\\39.7\end{pmatrix}$
	t_{tra}	65.2	33.8	441.9	438.2	216.4	38.4	133.7	1088.1
0.6	acc	0.7861	0.9747	0.9568	0.8471	0.9467	0.9039	0.9299	0.9792
	$\begin{pmatrix}N^1\\N^2\end{pmatrix}$	$\begin{pmatrix}103.5\\17.7\end{pmatrix}$	$\begin{pmatrix}92.7\\27.4\end{pmatrix}$	$\begin{pmatrix}10.4\\35.7\end{pmatrix}$	$\begin{pmatrix}11.3\\17.1\end{pmatrix}$	$\begin{pmatrix}27.7\\21.9\end{pmatrix}$	$\begin{pmatrix}19.6\\18.3\end{pmatrix}$	$\begin{pmatrix}28.0\\18.4\end{pmatrix}$	$\begin{pmatrix}40.5\\39.8\end{pmatrix}$
	t_{tra}	67.7	26.3	526.9	707.9	294.2	59.4	197.3	1533.4
0.7	acc	0.7841	0.9722	0.9573	0.8463	0.9475	0.9146	0.9333	0.9847
	$\begin{pmatrix}N^1\\N^2\end{pmatrix}$	$\begin{pmatrix}103.4\\16.4\end{pmatrix}$	$\begin{pmatrix}92.5\\35.6\end{pmatrix}$	$\begin{pmatrix}10.4\\34.1\end{pmatrix}$	$\begin{pmatrix}11.3\\22.5\end{pmatrix}$	$\begin{pmatrix}27.7\\20.9\end{pmatrix}$	$\begin{pmatrix}19.6\\19.6\end{pmatrix}$	$\begin{pmatrix}28.0\\21.6\end{pmatrix}$	$\begin{pmatrix}40.5\\41.3\end{pmatrix}$
	t_{tra}	70.8	27.7	1022.2	705.9	360.6	49.0	245.2	1795.3

It can be seen from Table S5 that ϵ_o can directly influence the accuracy and computational efficiency of the stacking ensemble model because it controls the dimensionality of the data after compression. A smaller ϵ_o will increase the computational efficiency of MECFNN because the data is compressed to a more compact form by the self-adaptive compressive projection (SACP) matrices. However, a smaller ϵ_o will potentially lead to a greater loss of information during the dimensionality compression, reducing the classification performance of MECFNN in terms of acc. While a greater ϵ_o will help MECFNN to preserve more information from data after dimensionality compression, it will decrease the system's computational efficiency.

Then, the influence of ρ_0 on the performance of MECFNN is investigated. In running the experiments, the value of ρ_0 is varied from 0.75, 0.85, 0.95 and 1. The other four externally controlled parameters follow the recommended setting, namely, $L = 2$, $W_o = 3W$, $\epsilon_o = 0.5$ and $\delta_o = e^{-3}$. The results obtained by MECFNN models with different ρ_0 are reported in Table S6 in terms of acc, N and t_{tra}. Note that the adaptive activation control scheme stops functioning with $\rho_0 = 1$ because all the IF-THEN rules will be recognised as being activated with respect to the input data by Condition 2.

Table S6 shows that ρ_0 has a greater impact on the computational efficiency of MECFNN. This is because ρ_0 helps Condition 2 to dynamically select these activated IF-THEN rules with respect to the current inputs for consequent parameter updating and output generation. A greater ρ_0 enables more IF-THEN rules to be selected by Condition 2 at each learning cycle, and vice versa. Increasing the value of ρ_0 will increase the computational complexity of MECFNN because a greater number of IF-THEN rules are involved in the output generation with their consequent parameters being updated with respect to the current inputs, which can also lead to overfitting. However, if ρ_0 is set to be too small, the classification performance of MECFNN may be impacted because the adaptive activation control scheme may drop out too many IF-THEN rules to generate the outputs precisely because of the loss of information.

Finally, the influence of δ_0 on the performance of MECFNN is investigated. In running the experiments, the value of δ_0 is varied from e^{-3}, e^{-4}, e^{-5} and e^{-6}. The other four externally controlled parameters follow the recommended setting, namely, $L = 2$, $W_o = 3W$, $\epsilon_o = 0.5$ and $\rho_o = 0.95$. The results obtained by MECFNN models with different δ_0 are reported in Table S7 in terms of acc, N and t_{tra}.

Table S6. Classification performance of MECFNN with different settings of ρ_0

ρ_0	Meas.	NSLKDD	CICIDS2017	MNIST	FMNIST	SIRIWHU	RSSCN		PatternNet
							1:4	1:1	
0.75	acc	0.7925	0.9733	0.9601	0.8306	0.9485	0.9166	0.9127	0.9818
	$\binom{N^1}{N^2}$	$\binom{103.4}{18.9}$	$\binom{92.6}{30.2}$	$\binom{10.4}{34.8}$	$\binom{11.3}{19.2}$	$\binom{27.8}{19.4}$	$\binom{19.7}{21.0}$	$\binom{27.9}{18.5}$	$\binom{40.5}{34.5}$
	t_{tra}	54.6	26.4	468.5	190.1	147.0	24.0	92.7	789.0
0.85	acc	0.7918	0.9743	0.9592	0.8462	0.9440	0.9169	0.9306	0.9824
	$\binom{N^1}{N^2}$	$\binom{103.4}{21.8}$	$\binom{92.6}{34.2}$	$\binom{10.4}{36.6}$	$\binom{11.3}{22.2}$	$\binom{27.6}{20.4}$	$\binom{19.6}{20.3}$	$\binom{28.0}{19.6}$	$\binom{40.5}{35.6}$
	t_{tra}	55.5	29.5	472.8	297.1	139.4	26.9	107.5	881.8
0.95	acc	0.7883	0.9743	0.9570	0.8480	0.9454	0.9165	0.9316	0.9840
	$\binom{N^1}{N^2}$	$\binom{103.5}{20.3}$	$\binom{92.8}{29.2}$	$\binom{10.4}{30.9}$	$\binom{11.3}{20.0}$	$\binom{27.8}{21.2}$	$\binom{19.7}{18.5}$	$\binom{28.0}{19.6}$	$\binom{40.5}{39.7}$
	t_{tra}	65.2	33.8	441.9	438.2	216.4	38.4	133.7	1088.1
1	acc	0.7824	0.9738	0.9556	0.8482	0.9429	0.9134	0.9317	0.9805
	$\binom{N^1}{N^2}$	$\binom{103.4}{11.3}$	$\binom{92.8}{25.7}$	$\binom{10.5}{26.4}$	$\binom{11.3}{17.3}$	$\binom{27.6}{19.9}$	$\binom{19.6}{19.7}$	$\binom{28.0}{17.8}$	$\binom{40.5}{48.9}$
	t_{tra}	148.7	90.4	673.5	619.2	401.4	63.2	299.7	2579.7

Table S7. Classification performance of MECFNN with different settings of δ_0

δ_0	Meas.	NSLKDD	CICIDS2017	MNIST	FMNIST	SIRIWHU	RSSCN		PatternNet
							1:4	1:1	
e^{-3}	acc	0.7883	0.9743	0.9570	0.8480	0.9454	0.9165	0.9316	0.9840
	$\binom{N^1}{N^2}$	$\binom{103.5}{20.3}$	$\binom{92.8}{29.2}$	$\binom{10.4}{30.9}$	$\binom{11.3}{20.0}$	$\binom{27.8}{21.2}$	$\binom{19.7}{18.5}$	$\binom{28.0}{19.6}$	$\binom{40.5}{39.7}$
	t_{tra}	65.2	33.8	441.9	438.2	216.4	38.4	133.7	1088.1
e^{-4}	acc	0.7815	0.9749	0.9584	0.8285	0.9335	0.9038	0.8620	0.9728
	$\binom{N^1}{N^2}$	$\binom{85.2}{12.4}$	$\binom{66.6}{17.9}$	$\binom{4.6}{18.0}$	$\binom{3.3}{14.7}$	$\binom{16.6}{12.6}$	$\binom{10.7}{12.1}$	$\binom{13.8}{13.0}$	$\binom{24.3}{19.5}$
	t_{tra}	44.0	25.5	130.6	153.5	90.0	15.4	52.6	452.5
e^{-5}	acc	0.7839	0.9712	0.9587	0.8059	0.7708	0.7399	0.7312	0.6701
	$\binom{N^1}{N^2}$	$\binom{60.0}{5.9}$	$\binom{50.9}{9.3}$	$\binom{2.9}{11.8}$	$\binom{2.1}{8.1}$	$\binom{12.2}{8.4}$	$\binom{5.9}{8.2}$	$\binom{7.7}{9.2}$	$\binom{14.3}{13.8}$
	t_{tra}	13.4	4.6	71.2	86.1	46.3	10.3	30.3	234.1
e^{-6}	acc	0.7842	0.9737	0.9596	0.7937	0.5977	0.6022	0.5131	0.5380
	$\binom{N^1}{N^2}$	$\binom{45.9}{5.3}$	$\binom{40.8}{6.3}$	$\binom{2.2}{7.6}$	$\binom{2.1}{6.9}$	$\binom{8.6}{7.0}$	$\binom{4.4}{6.5}$	$\binom{6.2}{7.3}$	$\binom{9.7}{9.8}$
	t_{tra}	10.8	4.3	71.2	85.1	34.9	6.5	19.1	196.4

One can see from Table S7 that, a greater δ_o enables MECFNN to identify more IF-THEN rules from data because it increases the sensitivity of the stacking ensemble model towards these more distinctive input samples that are spatially distant from existing prototypes. On the other hand, a smaller δ_o will reduce the size of the IF-THEN rule base learned from data because it forces MECFNN to focus on major data patterns. In general, a greater δ_o can help MECFNN to better capture the underlying patterns and structure of data with a larger IF-THEN rule base, but it will increase the computational complexity and lead to overfitting potentially. However, if δ_o is too small, the learned IF-THEN rules may be insufficient to approximate the problem precisely, and the classification performance of MECFNN will be influenced as a result.

E. Parameter Settings for Comparative Approaches

In running the numerical experiments, the number of nearest neighbours, k is set to be 3 for kNN, a commonly chosen setting by exiting works [1]. The maximum depth is set to be $K-1$ for DT to allow the tree structure to fully grow (K is the total amount of labelled data samples presented to DT). SVM uses the linear kernel, and the box constraint is 1 [2]. MLP has a four layer architecture with two hidden layers, each of which has 128 neurons. Root mean squared propagation (RMSprop) is employed as the optimiser with the learning rate and weight decay set as 0.001 and 0.9, respectively. RNN follows the recommended setting given by [3], which has one hidden layer composed of 80 recurrent neurons. The Adam optimiser utilised for tuning the hyperparameters of RNN. The learning rate is set to be 0.5 for intrusion detection problems and 0.005 for image classification problems. The LSTM follows the same parameter setting and architecture as the RNN except that the recurrent neurons are replaced by LSTM neurons. As [3] does not give the exact setting of batch size, the batch sizes for MLP, RNN and LSTM are all set as 50 in this study. BAT follows the exact same setting as [4] except that the kernel size for 1D convolutional layers in BAT set as 4 due to the lack of precise information in the original literature. All the four ANN models are trained for 100 epochs [4]. PALM [5] uses first-order fuzzy rules and local updating strategy. Its parameters are set as: $a=0.1,\ b_1=0.0002,\ b_2=0.01, c_1=0.01$ and $c_2=0.01$. SC follows the recommended setting given by [6]. The parameters of LREC are set as $b_1=0.85,\ b_2=0.2$, same as the experimental setting used in [7]. SOFBIS follows the recommended setting given by [8]. The externally controlled parameters of SEFIS are determined as: $K=0.5$; $\delta_1=0.5$; $\delta_2=0.5$, and $p_0=2$. For the 11 ensemble classifiers employed for the performance comparison, ADBDT, SAMDT and RF are composed of 50 DTs with the maximum depth set as $K-1$. ADBNN and SAMNN are also composed of 50 base kNN classifiers with k set as 3. The number of DTs in XGBoost is set as 40, and the maximum depth of each DT is 40. eEnsemble is composed of 10 eClass0 classifiers, and other parameters follow the same setting used in [9]. SOFEns, FWADB, STHPEF and MEFNN follow the respective recommended parameter settings given by [10]–[13]. Note that, similar to MECFNN and the four ANN models, MEFNN is trained for 100 epochs as well to maximise its classification performance.

F. Additional Numerical Results

Table S9. *p*-values returned from pairwise Wilcoxon signed rank tests

MECFNN versus	*p*-value	MECFNN versus	*p*-value
kNN	0.0006	SEFIS	0.0000
DT	0.0001	ADBDT	0.0014
SVM	0.1361	ADBNN	0.0006
MLP	0.4114	RF	0.0137
RNN	0.0008	SAMDT	0.0003
LSTM	0.0332	SAMNN	0.0001
BAT	0.0000	eEns	0.0000
PALM	0.0056	XGB	0.0777
SC	0.0029	SOFEns	0.0000
LREC	0.0039	FAWADB	0.0089
SOFBIS	0.0000	MEFNN	0.0014

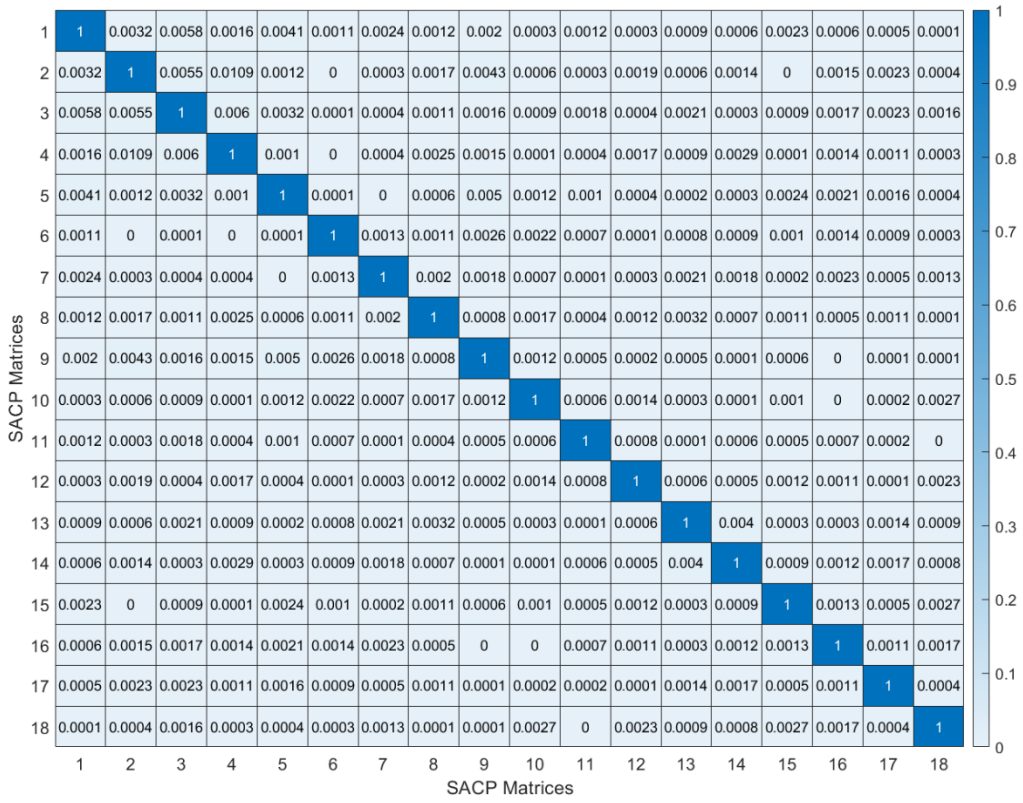


Fig. S1. Absolute correlation coefficients between SACP matrices associated with the learned IF-THEN rules in the first layer of MECFNN

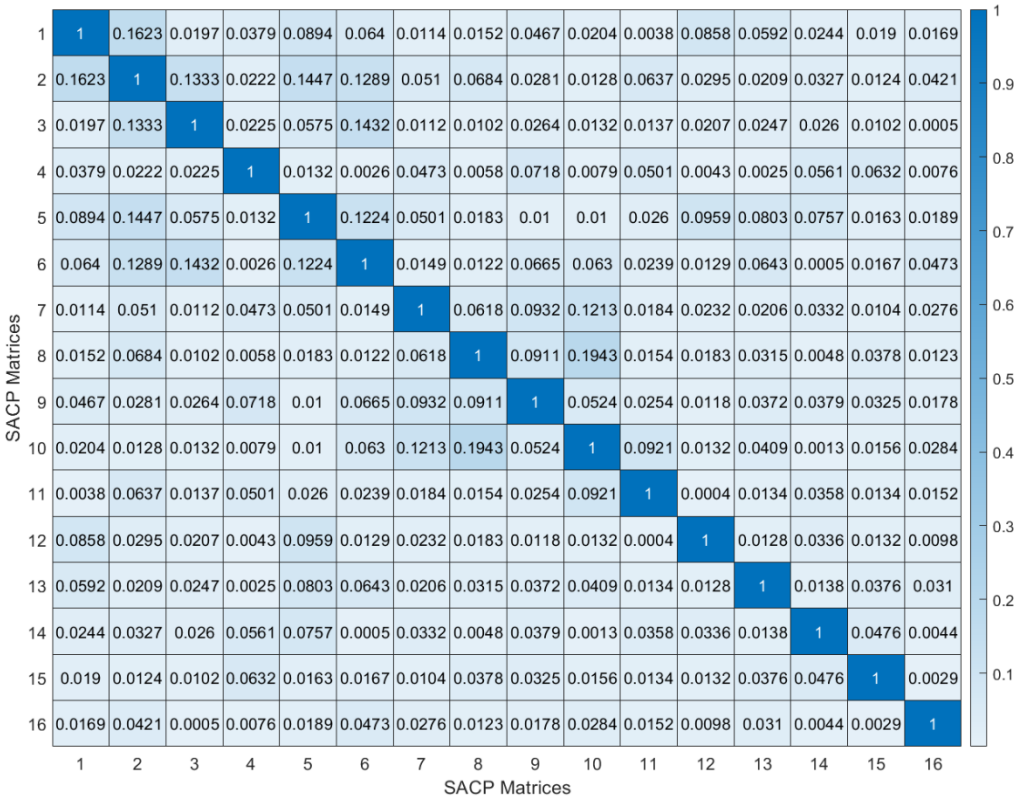


Fig. S2. Absolute correlation coefficients between SACP matrices associated with the learned IF-THEN rules in the second layer of MECFNN

G. Ablation Analysis

In this section, an ablation analysis is performed to demonstrate the influence of the adaptive dimensionality compression scheme and adaptive activation control scheme on the performance of MECFNN. In this example, three alterative versions of MECFNN are implemented, which include:

- Version 1 (denoted by $\text{MECFNN}_{\text{v1}}$): MECFNN but with adaptive dimensionality compression scheme implemented only (equivalent to setting $\rho_0 = 1$);
- Version 2 (denoted by $\text{MECFNN}_{\text{v2}}$): MECFNN but with adaptive activation control scheme implemented only;
- Version 3 (denoted by $\text{MECFNN}_{\text{v3}}$): MECFNN with both schemes implemented, but the compressive projection matrices are not updated during the learning process.

The performances of MECFNN and its three alternative versions are evaluated on the same 15 datasets under the same experimental protocols and parameter settings used in numerical examples presented in Tables I-III. In addition, MEFNN is also involved in this ablation analysis as the vanilla version of MECFNN.

The average acc rates, average numbers of rules per layer and the average training time costs per iteration (t_{tra}, in seconds) obtained by MECFNN, $\text{MECFNN}_{\text{v1}}$, $\text{MECFNN}_{\text{v2}}$, $\text{MECFNN}_{\text{v3}}$ and MEFNN on the 15 datasets under 23 different experimental settings are presented in Figs. S3-S5, respectively. The acc rates of the five stacking ensemble models per dataset per setting are tabulated in Table S10.

It is shown in Fig. S3 that MECFNN outperforms all its alternatives as well as its predecessor on the 15 datasets in terms of average acc. In particular, one can see from Table S10 that MECFNN achieves the best performance in 14 out of 23 cases, and outperforms its predecessor in 18 out of 23 cases. The experimental results demonstrate clearly the effectiveness of the proposed adaptive dimensionality compression and adaptive activation control schemes in enhancing the multi-level latent representation learning capability of MECFNN.

Comparing between $\text{MECFNN}_{\text{v1}}$ and MEFNN, from Figs. S3-S5 one can see that the adaptive dimensionality compression scheme enables the model to learn more discriminative features from data with less IF-THEN rules, at the cost of increased computational complexity and a greater susceptibility to overfitting. Comparing between $\text{MECFNN}_{\text{v2}}$ and MEFNN, it can be concluded that the adaptive activation control scheme enhances the computational efficiency and helps reduce overfitting. The combination of the two schemes effectively improves the performance of MECFNN without increasing the computational complexity significantly.

It is also worth noting that $\text{MECFNN}_{\text{v3}}$ performs the worst among the five stacking ensemble models. The main reason, as mentioned in Section I, is that the randomly generated VSRP matrices may lead to a significant loss of information, particularly when the compression ratio is high.

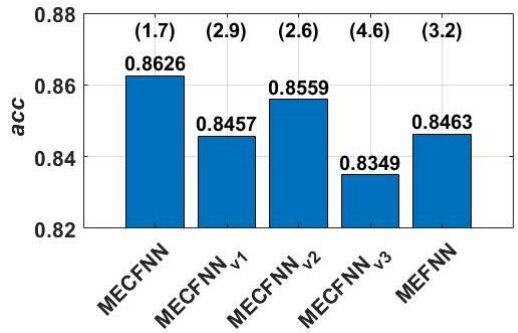


Fig. S3. Average classification accuracy rates of the five stacking ensemble models

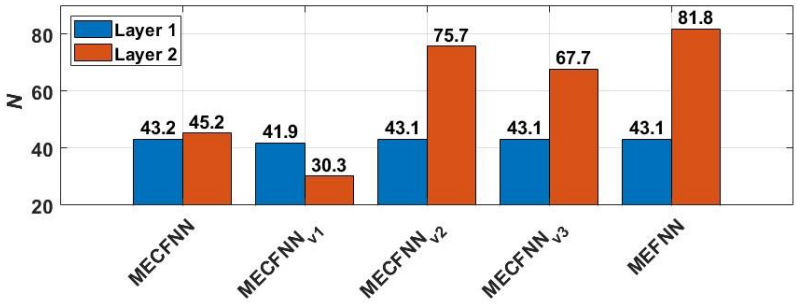


Fig. S4. Average numbers of rules per layer of the five stacking ensemble models

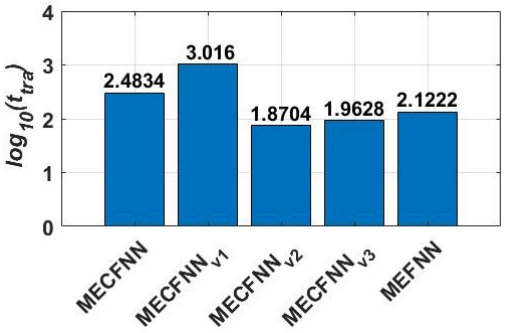


Fig. S5. Average training time costs per iteration of the five stacking ensemble models (logarithm is applied for visual clarity)

Table S10. Detailed ablation analysis results

Dataset		MECFNN	$MECFNN_{v1}$	$MECFNN_{v2}$	$MECFNN_{v3}$	MEFNN
NSLKDD		**0.7883**	0.7824	0.7830	0.7598	0.7756
UNSWNB15		**0.8485**	0.8454	0.8406	0.8225	0.8416
CICIDS2017		**0.9743**	0.9738	0.9710	0.9574	0.9722
CICIDS2018		**0.9848**	0.9840	0.9844	0.9719	0.9830
HIKARI2021		**0.9319**	0.9316	0.9314	0.9313	0.9317
MNIST	Orig	**0.9570**	0.9556	0.9503	0.9050	0.9499
	MBN	0.9724	0.7666	**0.9752**	0.9472	0.9733
	MBN2	0.9715	0.9721	**0.9742**	0.9452	0.9738
FMNIST	Orig	0.8480	0.8482	**0.8507**	0.8185	0.8486
	MBN	**0.8610**	0.7913	0.8542	0.8111	0.8519
	MBN2	0.8674	0.8491	**0.8705**	0.8171	**0.8705**
CIFAR10	MBN	**0.5069**	0.4955	0.5024	0.4220	0.5018
	MBN2	**0.5070**	0.4777	0.5037	0.4373	0.4996
SIRIWHU		0.9454	0.9429	**0.9475**	0.9415	0.9404
RSSCN7	1:4	**0.9165**	0.9134	0.9013	0.9011	0.9023
	1:1	0.9316	**0.9317**	0.9261	0.9247	0.9284
UCMerced	1:1	**0.9395**	0.9385	0.8984	0.9310	0.8421
	4:1	0.9612	0.9524	**0.9631**	0.9536	0.8814
AID	1:4	**0.9120**	0.9066	0.8804	0.8957	0.8683
	1:1	0.9336	0.9280	0.9376	0.9137	**0.9418**
PatternNet		**0.9840**	0.9805	0.9689	0.9688	0.9061
SAT4		**0.6921**	0.6783	0.6779	0.6668	0.6852
SAT6		0.6052	**0.6061**	0.5926	0.5606	0.5950

Finally, pairwise Wilcoxon signed rank tests [14] are carried out based on the acc rate per dataset (per experimental setting) to examine the statistical significance of the performance improvement of MECFNN over its three alternative versions and predecessor. The p-values returned from the pairwise tests are reported in Supplementary Table S11, where one can see that all the p-values are below the level of significance specified by $\alpha = 0.05$. This shows that the performance improvement is statistically significant.

Table S11. p-values returned from pairwise Wilcoxon signed rank tests for ablation analysis

MECFNN versus	p-value
$\text{MECFNN}_{\text{v1}}$	0.0002
$\text{MECFNN}_{\text{v2}}$	0.0042
$\text{MECFNN}_{\text{v3}}$	0.0000
MEFNN	0.0014

References

[1] A. A. Aburomman and M. Bin Ibne Reaz, "A novel SVM-kNN-PSO ensemble method for intrusion detection system," *Appl. Soft Comput. J.*, vol. 38, pp. 360–372, 2016.

[2] C. Zhang, J. Cheng, and Q. Tian, "Image-level classification by hierarchical structure learning with visual and semantic similarities," *Inf. Sci. (Ny).*, vol. 422, pp. 271–281, 2018.

[3] C. Yin, Y. Zhu, J. Fei, and X. He, "A deep learning approach for intrusion detection using recurrent neural betworks," *IEEE Access*, vol. 5, pp. 21954–21961, 2017.

[4] T. Su, H. Sun, J. Zhu, S. Wang, and Y. Li, "BAT: deep learning methods on network intrusion detection using NSL-KDD dataset," *IEEE Access*, vol. 8, pp. 29575–29585, 2020.

[5] M. M. Ferdaus, M. Pratama, S. G. Anavatti, and M. A. Garratt, "PALM: an incremental construction of hyperplanes for data stream regression," *IEEE Trans. Fuzzy Syst.*, vol. 27, no. 11, pp. 2115–2129, 2019.

[6] R. N. Patro, S. Subudhi, P. K. Biswal, and F. Dell'Acqua, "Dictionary-based classifiers for exploiting feature sequence information and their application to hyperspectral remotely sensed data," *Int. J. Remote Sens.*, vol. 40, no. 13, pp. 4996–5024, 2019.

[7] M. Ferdaus, B. Zhou, J. Wei, K. Lu, and J. Pan, "Significance of activation functions in developing an online classifier for semiconductor defect detection," *Knowledge-Based Syst.*, vol. 248, p. 108818, 2022.

[8] X. Gu, P. P. Angelov, and Q. Shen, "Self-organizing fuzzy belief inference system for classification," *IEEE Trans. Fuzzy Syst.*, vol. 30, no. 12, pp. 5473–5483, 2022.

[9] J. A. Iglesias, A. Ledezma, and A. Sanchis, "Ensemble method based on individual evolving classifiers," in *IEEE Conference on Evolving and Adaptive Intelligent Systems*, 2013, pp. 56–61.

[10] X. Gu, P. Angelov, and Z. Zhao, "Self-organizing fuzzy inference ensemble system for big streaming data classification," *Knowledge-Based Syst.*, vol. 218, p. 106870, 2021.

[11] X. Gu and P. Angelov, "Multi-class fuzzily weighted adaptive boosting-based self-organising fuzzy inference ensemble systems for classification," *IEEE Trans. Fuzzy Syst.*, vol. 30, no. 9, pp. 3722–3735, 2022.

[12] X. Gu, C. Zhang, Q. Shen, J. Han, P. P. Angelov, and P. M. Atkinson, "A self-training hierarchical prototype-based ensemble framework for remote sensing scene classification," *Inf. Fusion*, vol. 80, pp. 179–204, 2022.

[13] X. Gu, P. Angelov, J. Han, and Q. Shen, "Multilayer evolving fuzzy neural networks," *IEEE Trans. Fuzzy Syst.*, p. DOI: 10.1109/TFUZZ.2023.3276263, 2023.

[14] M. Friedman, "A comparison of alternative tests of significance for the problem of m rankings," *Ann. Math. Stat.*, vol. 11, no. 1, pp. 86–92, 1940.