CUBEDIMAE: Automatic Autoencoder Generation based on Dimension Estimation by Tessellation

Byungrok Kim
KENTECH
Naju, South Korea
fried4chicken@kentech.ac.kr

Myeong-Ha Hwang
KEPRI
Daejeon, South Korea
mh.hwang@kepco.co.kr

Jeonghyun Joo
KEPRI
Daejeon, South Korea
jh.joo590@kepco.co.kr

YooJin Kwon
KEPRI
Daejeon, South Korea
kwon43@kepco.co.kr

Hyunwoo Lee
KENTECH
Naju, South Korea
hwlee@kentech.ac.kr

***Abstract*—Due to the difficulty of getting the high-quality labels required for supervised learning, unsupervised learning has received considerable attention in diverse domains. One of the recent and widely used unsupervised algorithms is the autoencoder, but designing an optimal autoencoder for a specific dataset is challenging. In detail, an autoencoder uses dimensionality reduction to learn data distribution with the bottleneck, and it is challenging to find the optimal number of nodes in the bottleneck, which is closely related to the intrinsic dimension.**

We propose CUBEDIMAE, a framework that estimates the intrinsic dimension and designs an optimal autoencoder. It approximates data in an ingenious way to emulate continuity to build on the idea of the degree of freedom to estimate the dimension. To show its feasibility, we implement a proof-of-concept of our method and release the source code. Our numerical result shows that it works correctly on 5 sets of data and efficiently generates optimal autoencoders, saving 41.2% in time compared with the baseline approach.

I. INTRODUCTION

There has been a considerable advance in artificial intelligence (AI) technology, which contributes to making systems automated and efficient in many domains. However, unsupervised learning has been paid much attention because getting high-quality labels for supervised learning is difficult.

An autoencoder is one of the widely used unsupervised learning algorithms, which learns data distribution by compressing the data (encoder) and then decompressing it (decoder). Due to its simplicity, there have been a lot of applications in diverse domains, including security [1], [2], market prediction [3], fault diagnosis [4], and image compression [5].

However, finding an optimal autoencoder for a specific dataset is challenging. An autoencoder, which learns the distribution through dimensionality reduction, fails to reconstruct the input if the latent dimension is less than a certain threshold, which puts an upper bound on the compression ratio. Identifying such a threshold is helpful, as we can make a lightweight autoencoder with minimal latent dimension. It has been observed that the threshold matches the intrinsic dimension of input data [6].

There have been many attempts [6]–[8] to identify the intrinsic dimension by training an autoencoder multiple times, adjusting the latent dimension, and observing the reconstruction performance. However, current gradient descent algorithms depend on stochastic processes, producing different results on every trial even if the optimal value of the latent dimension has been found by observing the reconstruction losses.

To address such a challenge, in this paper, we propose CUBEDIMAE, a framework to identify the intrinsic dimension in an efficient and deterministic way and generate an autoencoder automatically. As datasets are discontinuous and dimension concepts assume continuity, we suggest tessellation to emulate continuity. Then, our framework finds the dataset's intrinsic dimension. Finally, a simple autoencoder is generated, whose latent dimension is identified as the intrinsic dimension.

To show its feasibility, we implement CUBEDIMAE and evaluate it by identifying the intrinsic dimensions of 5 datasets, including the s-curve, the Swiss roll, and spheres. The experiments show that CUBEDIMAE estimates the dimensions efficiently, showing 41.2% reduction in time, compared with the baseline approach [6] that relies only on repeated training to find the optimal configuration.

In summary, we make the following contributions:

- We design CUBEDIMAE, a framework that identifies the intrinsic dimension by tessellation and automatically generates an autoencoder based on the intrinsic dimension.
- We implement CUBEDIMAE and release our source code on the public repository at https://github.com/sandwich-coder/CubeDimAE.
- We demonstrate the feasibility of CUBEDIMAE on 5 datasets and show its efficiency compared with the baseline approach.

II. BACKGROUND

This section provides background knowledge of the intrinsic dimension, autoencoder, and tessellation because we are going to find the *intrinsic dimension* by *tessellation* and use it as the latent dimension of an *autoencoder*.

2375-9356/25/$31.00 ©2025 IEEE
DOI 10.1109/BigComp64353.2025.00013

A. Intrinsic Dimension

The **intrinsic dimension** refers to the minimum number of parameters needed to provide a good approximation of a dataset [9]–[13]. However, there is no rigorous mathematical definition of the quantity. Instead, there is a concept called *inductive* dimension, where the dimension of an object is determined by that of its boundary. According to the inductive dimension, the dimensions of a curve and a surface in a Euclidean space are one and two, respectively. The inductive dimension of a point is 0 and its boundary, which is an empty set, has the dimension of -1. [14] The research community has focused on finding the intrinsic dimension to identify the lower bound for the dimensionality reduction or to measure the complexity [15].

B. Autoencoder

An **autoencoder** is a neural network that compresses and then reconstructs the input. Its predecessor, principal component analysis (PCA), has proved highly effective in linear dimensionality reduction through linear coordinate transformation but performs poorly on nonlinear distributions. As an alternative, autoencoders were proposed for learning nonlinear coordinate transformations to compress nonlinear datasets adequately [15].

An autoencoder consists of an *encoder* and a *decoder*. It has a layer called *bottleneck* and the number of nodes in the bottleneck is called *latent dimension*. The bottleneck is a layer with the smallest number of nodes and is the output of the encoder. The encoder compresses the dataset and the decoder reconstructs it. Then, the model is trained to reconstruct the input with as small a loss as possible. The compression restricts the reconstruction to be effective only on the trained dataset, resulting in large discrepancies on data whose distribution differs from the trained. Such capability of "learning" data distribution resulted in wide adoption in various areas, ranging from noise reduction to data compression [16]–[19].

C. Tessellation

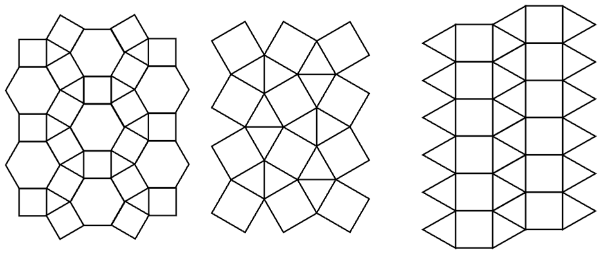


Figure 1: 2-dimensional tessellations

A **tessellation**, or *tiling*, is the covering of a space with geometric shapes with no gap nor overlap [20]. The shapes that cover a space are called 'tiles.' It is equivalent to partitioning a space in that the separated regions can be considered individual tiles (see Figure 1).

Instead of using any shapes, tiling with identical regular shapes is called a regular tessellation. Regular tessellations are used where each tile should cover the same amount of area in a structured manner, such as the sensors in a camera that receive light or tiles that constitute a typical floor. It has been proved that hypercube is the only type of regular polytope that can cover an entire Euclidean space of arbitrary dimension. [20]

III. CUBEDIMAE

This section presents a framework, called CUBEDIMAE (Figure 2), that aims to design an optimal autoencoder.

Approach. CUBEDIMAE takes a dataset and automatically outputs an autoencoder. Our approach is to set the latent dimension as the intrinsic dimension of the input.

Challenge. The most challenging is identifying the intrinsic dimension, because the concepts of dimension we are familiar with assume continuity. Recall that the dimension of a set of discontinuous points is 0 according to the inductive dimension.

Solution. Instead of creating a new concept of dimension, we express the data in a form that emulates continuity. It consists of three steps:

- **Tiling:** first, we tessellate the Euclidean space with identical tiles.
- **Representation:** next, we "fill" the tiles containing at least one data instance. We call the set of the "filled" a representation of the data.
- **Estimation:** finally, we compute the intrinsic dimension based on the representation.

Problem scope. With the identified dimension, CUBEDIMAE generates an autoencoder. All layers except the bottleneck are fixed and the algorithm finds the adequate latent dimension, putting the problem of automatic design of the rest as future work.

IV. DETAIL OF CUBEDIMAE

This section details the research challenges and our approach in each step of CUBEDIMAE.

A. Tiling

We partition the entire space, $\mathbb{R}^n$, with an n-dimensional square grid. Then, the separated regions are tiles that cover the space. Here, the tile size affects how a dataset would be represented (see Figure 3). It is necessary to decide the appropriate size to get the correct intrinsic dimension, so we parameterize the tile size:

Definition 1 (*ℓ-tile*). We call a tile having an edge length of ℓ an ℓ-tile.

Instead of fixing the length, the framework finds the appropriate minimum and maximum for the data distribution. Once the range is set, the estimator iterates calculations over the range to find the intrinsic dimension.

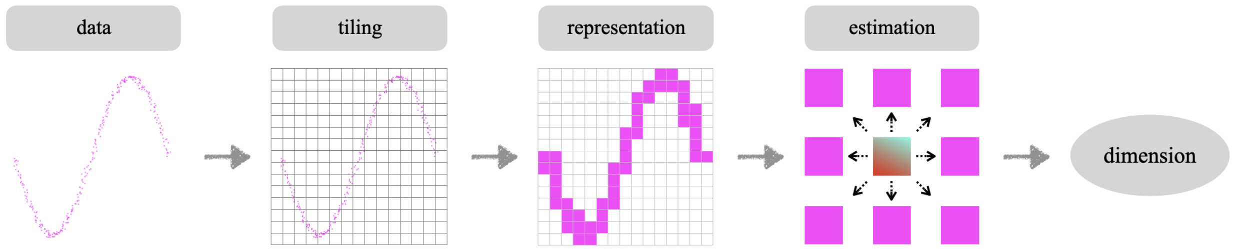


Figure 2: **Overview**

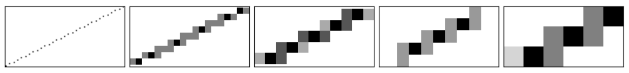


Figure 3: Representations of a line with different tile sizes

To indicate a specific tile, we index it according to its position relative to the origin. Let $\mathcal{T}$ be a set of tiles in an n-dimensional space and x be an element. Then, x is expressed as a pair of integers, a position relative to the one at the origin.

$$\mathrm{x} = (x_1, x_2, x_3, \cdots, x_n) \tag{1}$$

In a 2-dimensional space for example, a tile whose position is (a, b) is an ath tile along the first axis and bth along the second axis, counting from the tile at the origin. This way of indexing leads to

$$\mathcal{T} = \mathbb{Z}^n. \tag{2}$$

From this equivalence, we can define the notion of two tiles being next to each other.

Definition 2 (*adjacent*)**.** Let $\mathcal{T}$ be the set of n-dimensional tiles. For two tiles $\mathrm{x}, \mathrm{y} \in \mathcal{T}$, expressed as $(x_1, x_2, \cdots, x_n)$ and $(y_1, y_2, \cdots, y_n)$ respectively, they are said to be **adjacent** to each other if

$$\max_n |x_n - y_n| = 1. \tag{3}$$

In other words, two tiles are adjacent if their Chebyshev distance equals 1 [21]. This is a mathematical formulation of two n-dimensional tiles sharing at least one vertex.

B. Representation

The second step is representation. Our approach is to "fill" every tile containing at least one data instance. With this approach, we obtain a representation covering discontinuous points in a coarse-grained way (see Figure 4).

Definition 3 (*occupied*)**.** We call a tile that contains data 'occupied'.

Definition 4 (*vicinity*)**.** For an occupied tile, its **vicinity** is the number of adjacent tiles that are *occupied*. The vicinity of an empty tile is undefined.

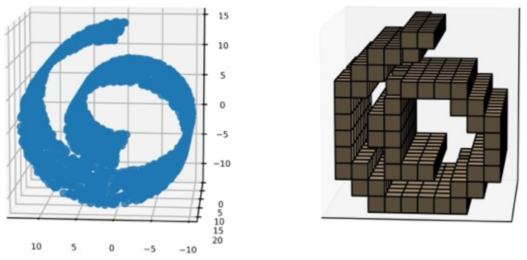


Figure 4: Representation of the Swiss roll

C. Estimation

To find the intrinsic dimension, we compute the average vicinity of all the occupied. The amounts of data contained are added as weights to the average to reduce the effect of outliers. The calculation is repeated for different tile sizes, ℓ $(m \leq \ell \leq M)$. Lastly, we find the ℓ at which the average vicinity reaches the maximum and get the intrinsic dimension from the representation at that length. A representation with the maximum mean vicinity shows the distribution in the *most connected* manner. The dimension is obtained by the following theorem.

Theorem 1 (*saturation vicinity*)**.** *If all tiles in an n-dimensional space are occupied, the vicinity of every tile is* $3^n - 1$.

The above theorem concerns a hypothetical scenario in which a dataset "saturated" the entire n-dimensional space so that every tile contained some data. We prove the theorem and show how the idea of saturation eventually leads to dimension estimation.

Proof. Consider a set of tiles, $\mathcal{T}$, which is $\mathbb{Z}^n$, and a tile $\mathrm{x} = (x_1, x_2, \cdots, x_n) \in \mathcal{T}$. Let S_n be the set containing the x and all its adjacent elements. Below we show that the size of S_n is 3^n.

$$|S_n| = \left| \left\{ \mathrm{x}' \,\middle|\, \max_n |x'_n - x_n| = 1 \right\} \cup \{\mathrm{x}\} \right| = 3^n$$

As the distance to x from itself is 0, $\max_n |x_n - x_n| = 0$. Thus, S_n can be re-written as:

$$S_n = \left\{ \mathrm{x}' \,\middle|\, \max_n |x'_n - x_n| \leq 1 \right\}.$$

Let $\beta[x]$ as the set of 1-dimensional tiles and all its adjacent ones.

$$\beta[x] := \{x-1,\, x,\, x+1\} \quad (x \in \mathbb{Z}).$$

Then, S_1 is $\beta[x]$ as follows:

$$\begin{aligned} S_1 &= \{x' \mid \max|x'-x| \le 1\} \\ &= \{x' \,||x'-x| \le 1\} \\ &= \{x-1,\, x,\, x+1\} \\ &= \beta[x]. \end{aligned}$$

Definitely, $|S_1| = |\{x-1, x, x+1\}| = 3 = 3^1$. Let $k \in \mathbb{N}$. Assume $|S_k| = 3^k$. Then,

$$\begin{aligned} S_{k+1} &= \left\{ \mathrm{x}' \,\middle|\, \max_{k+1} |x'_{k+1} - x_{k+1}| \le 1 \right\} \\ &= \left\{ \mathrm{x}' \,\middle|\, \max\left(\max_k |x'_k - x_k|\,,\, |x'_{k+1} - x_{k+1}| \right) \le 1 \right\} \\ &= \left\{ \mathrm{x}' \,\middle|\, \max_k |x'_k - x_k| \le 1,\, |x'_{k+1} - x_{k+1}| \le 1 \right\} \\ &= \left\{ \mathrm{x}' \,\middle|\, (x'_1, \cdots, x'_k) \in S_k,\, x'_{k+1} \in \beta[x_{k+1}] \right\} \\ &= S_k \times \beta[x_{k+1}], \end{aligned}$$

which leads to $|S_{k+1}| = |S_k \times \beta[x_{k+1}]| = |S_k| 3 = 3^{k+1}$. From this mathematical induction, we get

$$|S_n| = 3^n.$$

It shows that an element in $\mathbb{Z}^n$ always has $3^n - 1$ *adjacent* elements. Therefore, the vicinity of every tile is $3^n - 1$ if all the tiles are occupied. □

Now, we can do a thought experiment. Suppose all tiles in a k-dimensional space are occupied. Then, add an axis orthogonal to the space to extend the dimension. For the occupied, the vicinity does not change because every such tile still has the vicinity of $3^k - 1$. It is not $3^{k+1} - 1$, because it was *contained* in a k-dimensional space.

This conclusion sheds light on a key feature of dimension. When we call a plane in a 3-dimensional space 2-dimensional, it can be interpreted that the plane could be *contained* in some 2-dimensional space. Likewise, if the representation of a dataset in an n-dimensional space has a vicinity of $3^k - 1$ $(k \le n)$ on average, we can interpret the set of tiles as those that would fully occupy and could be contained in a k-dimensional space. From this observation, we conclude a dataset is n-dimensional if the vicinity, for the representation, averages to $v = 3^n - 1$. In detail, we finally get the intrinsic dimension, dim, from v as follows:

$$\mathrm{dim} = \mathrm{round}\left(\log_3(v+1)\right),$$

where round is the round-half-up function.

V. Evaluation

This section evaluates CubeDimAE on two measures – accuracy and efficiency.

Layer	**# of Nodes**	**Activation**
input	3	
1	100	GELU
2	99	GELU
3	98	GELU
4	97	GELU
5	96	GELU
latent	***k***	GELU
6	96	GELU
7	97	GELU
8	98	GELU
9	99	GELU
10	100	GELU
output	3	sigmoid

Table I: Network configuration

Dataset	**Estimated** (vicinity)	**Answer**	**Correct?**
S-curve	2 (11.133)	2	✓
Swiss roll	2 (10.916)	2	✓
Möbius strip	2 (10.870)	2	✓
Hollow sphere	2 (11.943)	2	✓
Solid sphere	3 (22.413)	3	✓

Table II: **Accuracy**

A. Experiment Setting

Implementation. We implement CubeDimAE by Python 3.11.0 using numpy, scikit-learn, and keras.

Target dataset. For the evaluation of CubeDimAE, we generate 5 artificial datasets, which are typical examples of nonlinearity. (see Figure 5):

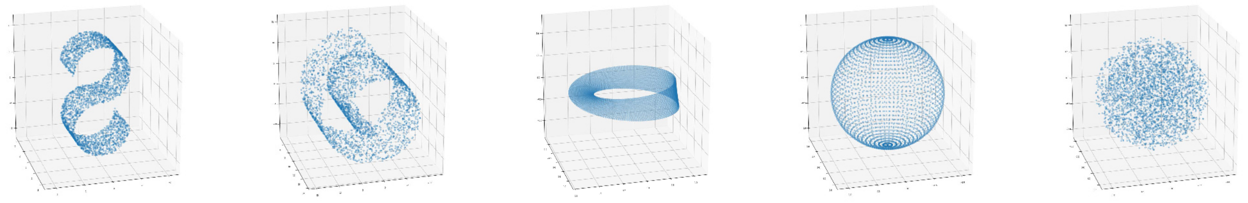
- **S-curve & Swiss roll:** datasets widely used to test nonlinear dimensionality reduction techniques, such as locally linear embedding (LLE) or multidimensional scaling (MDS). Both the s-curve and the Swiss roll are 2-dimensional.
- **Möbius strip:** a Möbius strip of radius 1 and width 0.5. It is 2-dimensional.
- **Hollow sphere:** a mesh grid of uniform distributions of 80 points along the azimuth and 40 along the elevation. It is 2-dimensional.
- **Solid sphere:** a uniform solid sphere of radius 1. It is 3-dimensional.

Experiments for dimension identification. We compute the dimension at the maximum point of the average vicinity and then round it to the nearest integer. For the sake of analysis and visualization, the experiments consist only of datasets of 3 parameters.

Auto-generation of an autoencoder. We design an autoencoder based on the intrinsic dimension identified. The network configuration is shown in Table I.

B. Accuracy

We show the accuracy of CubeDimAE by comparing the outputs with the answers. The results show that CubeDimAE correctly works for all the datasets (see Table II).



(a) S-curve (2-dimensional) (b) Swiss roll (2-dimensional) (c) Möbius strip (2-dimensional) (d) Hollow sphere (2-dimensional) (e) Solid sphere (3-dimensional)

Figure 5: **Datasets**

To understand, we illustrate the process of CUBEDIMAE on the s-curve as a case study (Figure 6). We define the scale of a dataset as the maximum range of all the parameters. The edge length, ℓ, ranges from 1% to 10% of the scale. We calculate the average vicinities throughout the range with 1% interval. The average vicinities range from 2.13 to 11.06. At the maximum point, we get 2 as the intrinsic dimension referring to Theorem 1. Finally, CUBEDIMAE generates the neural network of latent dimension 2.

To verify the accuracy of the result, we train an autoencoder multiple times over the latent dimensions from 1 to 5 (i.e., the k in Table I). The autoencoder with $k = 1$ shows a higher loss than other cases (see Figure 6), and those with $k \geq 2$ show similar losses. This means 2 is the minimum latent dimension that enables an autoencoder to learn the s-curve distribution adequately.

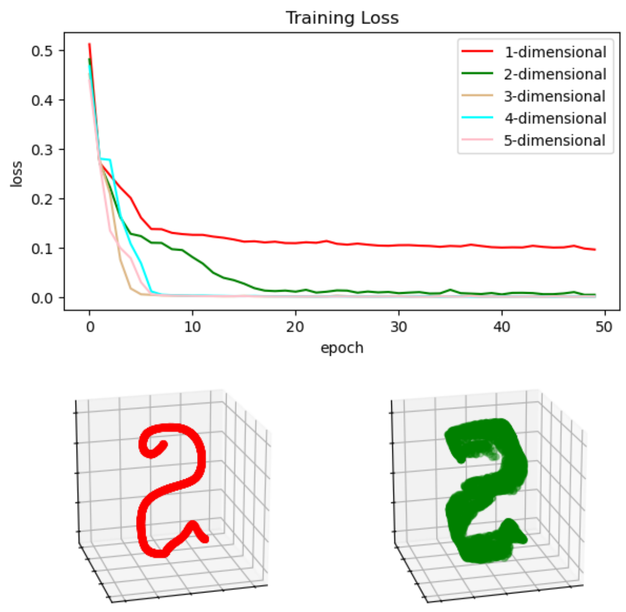


Figure 6: S-curve case study

We illustrate the reconstructions of the s-curve with $k = 1$ and $k = 2$ (see Figure 6). The 1-dimensional reconstruction, the red figure on the left, only follows the "middle line" of the s-curve, rather than fully threading all parts of the surface. It can be observed that the s-curve compressed to two parameters is adequately reconstructed compared with that compressed to only one.

	Baseline (*s*)				**CUBEDIMAE (*s*)**		
Dataset	**AE1**	**AE2**	**AE3**	**Total**	**Est.**	**AE**	**Total**
S-curve	7.29	7.25	7.66	22.2	3.2	7.25	10.45
Swiss Roll	6.92	7.06	7.34	21.32	5.42	7.06	12.48
Möbius strip	6.94	7.08	7.32	21.34	3.03	7.08	10.11
Hollow sphere	7.06	7.07	7.35	21.48	5.67	7.07	12.74
Solid sphere	6.97	7.09	7.33	21.39	10.12	7.33	17.45

Table III: **Efficiency**

C. *Efficiency*

We show our CUBEDIMAE efficiently designs an optimal autoencoder for a given dataset. To this end, we consider the baseline approach [6], for the comparison with CUBEDIMAE. In the baseline approach an autoencoder is tested with all the possible latent dimensions, which are less than or equal to the dimensionality of data, and the losses are computed. In the experiment, we use the identical network configuration and the same hyper-parameters (e.g., the batch size or the epoch) for both approaches. For the metric, we measure the *elapsed time* to obtain an optimal autoencoder. The two approaches consist of:

- **Baseline:** the time spent training all the autoencoders. An optimal autoencoder has a latent dimension below which the reconstruction performance drops drastically.
- **CUBEDIMAE:** the time spent identifying the intrinsic dimension and training an autoencoder once.

We demonstrate the elapsed times in Table III. CUBEDIMAE shows a significant improvement in efficiency.

VI. RELATED WORK

Autoencoders to estimate the intrinsic dimension. Much research has been conducted to get the intrinsic dimension using autoencoders. Wang *et al.* [6] shows that there is a relationship between the intrinsic dimension of input data and the number of nodes in the compression layer and demonstrates its possibility on synthetic data and the MNIST dataset [22]. Bahadur *et al.* [7] uses autoencoders to identify the intrinsic dimension by leveraging sparsity-inducing penalties to control the number of non-zero nodes in the compression layer instead of changing the network structure. Similarly, Kärkkäinen *et al.* [8] use antoencoders with sparsity penalties and singular value proxies in identifying the intrinsic dimension. Note that the studies above focus on estimating the intrinsic dimension using autoencoders. On the contrary, our work aims to automatically generate an optimal autoencoder for a given dataset, assuming the number of nodes in the compression layer is closely related to the intrinsic dimension that the studies report.

Automatic generation of a neural network. There have been several studies to generate neural networks automatically. Assunçao *et al.* [23] use grammar-based genetic programming by using a grammar to encode network topology and parameters and making the topology and parameters evolve. The evolution of the network structure finally results in a reasonable performance. Safarik *et al.* [24] uses the genetic algorithm to find appropriate hyperparameters for a neural network. These works [25] mostly rely on evolutionary techniques. Unlike the existing works that utilize genetic algorithms to generate a network, we create an autoencoder with a deeper understanding of the concept of dimension.

VII. Conclusion

We propose CubeDimAE, a framework that identifies the intrinsic dimension of data and designs the network by itself. We demonstrate its feasibility on 5 datasets, showing 100% accuracy. We also report the efficiency of CubeDimAE compared with the baseline approach, showing 41.2% reduction in time in finding an optimal autoencoder. In future work, we will test CubeDimAE on real-world data, such as MNIST, to assess the efficacy. Further, we will augment our framework with outlier removal and noise separation mechanisms to improve the accuracy of dimension identification on complicated datasets. We will apply our approach to the autoencoder-based anomaly detection as the main application of our algorithm, as well as compression.

Acknowledgement

This research was supported by Korea Electric Power Corporation. (Grant number: R24XO01-3)

References

[1] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An ensemble of autoencoders for online network intrusion detection," in *Network and Distributed Systems Security (NDSS) Symposium*, 2018.

[2] F. Farahnakian and J. Heikkonen, "A deep auto-encoder based approach for intrusion detection system," in *2018 20th International Conference on Advanced Communication Technology (ICACT)*, pp. 178–183, IEEE, 2018.

[3] D. Mohanty, A. K. Parida, and S. S. Khuntia, "Financial market prediction under deep learning framework using auto encoder and kernel extreme learning machine," *Applied Soft Computing*, vol. 99, p. 106898, 2021.

[4] Z. Yang, B. Xu, W. Luo, and F. Chen, "Autoencoder-based representation learning and its application in intelligent fault diagnosis: A review," *Measurement*, vol. 189, p. 110460, 2022.

[5] L. Theis, W. Shi, A. Cunningham, and F. Huszár, "Lossy image compression with compressive autoencoders," in *International conference on learning representations*, 2022.

[6] Y. Wang, H. Yao, and S. Zhao, "Auto-encoder based dimensionality reduction," *Neurocomputing*, vol. 184, pp. 232–242, 2016.

[7] N. Bahadur and R. Paffenroth, "Dimension estimation using autoencoders with applications to financial market analysis," in *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pp. 527–534, IEEE, 2020.

[8] T. Kärkkäinen and J. Hänninen, "Additive autoencoder for dimension estimation," *Neurocomputing*, vol. 551, p. 126520, 2023.

[9] F. Camastra and A. Staiano, "Intrinsic dimension estimation: Advances and open problems," *Information Sciences*, vol. 328, pp. 26–41, 2016.

[10] P. Pope, C. Zhu, A. Abdelkader, M. Goldblum, and T. Goldstein, "The intrinsic dimension of images and its impact on learning," *arXiv preprint arXiv:2104.08894*, 2021.

[11] A. Ansuini, A. Laio, J. H. Macke, and D. Zoccolan, "Intrinsic dimension of data representations in deep neural networks," in *Advances in Neural Information Processing Systems* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), vol. 32, Curran Associates, Inc., 2019.

[12] E. Facco, M. d'Errico, A. Rodriguez, and A. Laio, "Estimating the intrinsic dimension of datasets by a minimal neighborhood information," *Scientific reports*, vol. 7, no. 1, p. 12140, 2017.

[13] K. Fukunaga and D. R. Olsen, "An algorithm for finding intrinsic dimensionality of data," *IEEE Transactions on computers*, vol. 100, no. 2, pp. 176–183, 1971.

[14] R. Engelking, *Dimension theory*, vol. 19. North-Holland Publishing Company Amsterdam, 1978.

[15] M. A. Kramer, "Nonlinear principal component analysis using autoassociative neural networks," *AIChE journal*, vol. 37, no. 2, pp. 233–243, 1991.

[16] L. Theis, W. Shi, A. Cunningham, and F. Huszár, "Lossy image compression with compressive autoencoders," in *International conference on learning representations*, 2022.

[17] F. Mentzer, E. Agustsson, M. Tschannen, R. Timofte, and L. Van Gool, "Conditional probability models for deep image compression," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4394–4402, 2018.

[18] J. Ballé, V. Laparra, and E. P. Simoncelli, "End-to-end optimized image compression," *arXiv preprint arXiv:1611.01704*, 2016.

[19] L. Gondara, "Medical image denoising using convolutional denoising autoencoders," in *2016 IEEE 16th international conference on data mining workshops (ICDMW)*, pp. 241–246, IEEE, 2016.

[20] H. Coxeter, *Regular polytopes*. Dover Publication, Inc, 1973.

[21] R. Coghetto, "Chebyshev distance," *Formalized Mathematics*, vol. 24, no. 2, pp. 121–141, 2016.

[22] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[23] F. Assunçao, N. Lourenço, P. Machado, and B. Ribeiro, "Automatic generation of neural networks with structured grammatical evolution," in *2017 IEEE congress on evolutionary computation (CEC)*, pp. 1557–1564, IEEE, 2017.

[24] J. Safarik, J. Jalowiczor, E. Gresak, and J. Rozhon, "Genetic algorithm for automatic tuning of neural network hyperparameters," in *Autonomous Systems: Sensors, Vehicles, Security, and the Internet of Everything*, vol. 10643, pp. 168–174, SPIE, 2018.

[25] E. Vonk, L. C. Jain, and R. P. Johnson, *Automatic generation of neural network architecture using evolutionary computation*, vol. 14. World Scientific, 1997.