PERFORMANCE COMPARISON OF SELF-ORGANIZING MAPS BASED ON DIFFERENT AUTOENCODERS

BY

BOREN ZHENG

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

COMPUTER SCIENCE

UNIVERSITY OF RHODE ISLAND

2019

MASTER OF SCIENCE THESIS

OF

BOREN ZHENG

APPROVED:

Thesis Committee:

Major Professor	Lutz Hamel
	Noah Daniels
	Frederick J. Vetter
	Nasser H. Zawia
	DEAN OF THE GRADUATE SCHOOL

UNIVERSITY OF RHODE ISLAND

2019

ABSTRACT

The Autoencoder (AE) is a kind of artificial neural network, which is widely used for dimensionality reduction and feature extraction in unsupervised learning tasks. Analogously, the Self-Organizing Map (SOM) is an unsupervised learning algorithm to represent the high-dimensional data by a 2D grid map, thus achieving dimensionality reduction. Some recent work has shown improvement in performance by combining the AEs with the SOMs. Knowing which variations of AEs work best and finding out whether the selection of AEs is data-depended or not is the purpose of this research.

Five types of AEs are implemented in this research; three different data sets are used for training; map embedding accuracy and estimated topographic accuracy are used for measuring the model quality. Overall, this research shows that nearly all AEs at least improve the SOM performance, improving embedding accuracy and letting the training process become efficient. The Convolutional Autoencoder (ConvAE) shows an outstanding performance in image-related data set, the Denoising Autoencoder (DAE) works well with the real-word data with noise, and the Contractive Autoencoder (CAE) performs excellently in the synthetic data set. Therefore, we can see that the selection of AEs depends on the properties of data.

ACKNOWLEDGMENTS

During the past two years of study at the University of Rhode Island, I would like to express the most profound appreciation to my advisor Dr. Hamel for his patient guidance and support throughout my academic study. I would also like to thank my committee members: Dr. Daniels and Dr. Vetter, for offering their time and suggestions. And thank Dr. Eaton for attending and chairing my defense.

I would also like to thank my friend Li Yuan and other staffs in the CS department for their inspiration and support.

A special thanks to my family – my parents and my aunts, for their constant love and encouragement. I would also like to express my thanks to my boyfriend Pengcheng, for his various help and ideas.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

The Autoencoder (AE) is a kind of artificial neural network. It is an unsupervised learning algorithm that is mainly used for feature extraction and dimensionality reduction [1]. It consists of an encoder and a decoder, which intend to reconstruct the original input data from the hidden layer representation. The architecture of an AE is shown in Figure 1.

The Self-Organizing Maps (SOMs) proposed by T. Kohonen [2] is another approach to reduce dimensionality, which shows the clustering results for high-dimensional input data onto a 2D grid map. In recent research, combining the AEs with SOMs has shown some promise in improving the performance of regular SOMs [3]. A Deep Neural Maps (DNMs) [4] model proposed in 2018 achieved this combination and gave excellent performance in high-dimensional data visualization. However, there are many different kinds of AEs, and knowing which one works best is an open question. Performance comparison of different AEs could help one find more appropriate AEs for a data set, hence improving the performance of the underlying SOMs.

In fields such as genomic data clustering [5] [6] and cluster analysis of massive astronomical data [7] [8], the SOM is a good approach since it does not only accomplish the clustering task but also provides an accessible visible clustering representation. However, because both genomic data and astronomical data are high-dimensional, it takes the SOM a long time to train the data. The AE is an excellent method to bring the data to a lower dimensionality while keeping the intrinsic structure of the data. Hence, the SOM in conjunction with AE could help save the training time. This project can help select an appropriate AE for a data set to reduce data dimensionality, thus reducing the computing time of SOM.

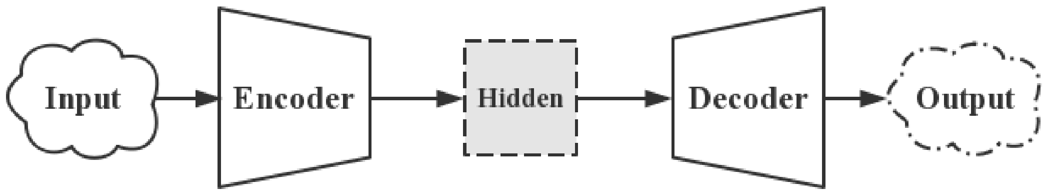


Figure 1. The architecture of an autoencoder (the shape of flowcharts does not represent the dimension variation)

In this research, I implemented five types of AEs which are basic Autoencoder (AE), Sparse Autoencoder (SAE), Contractive Autoencoder (CAE), Denoising Autoencoder (DAE), and Convolutional Autoencoder (ConvAE). I fed the SOM with the encoded data extracted from the five types of AEs and evaluated the performance in three different data sets. I selected a synthetic data set called ‘dim064’, a real-word data set named ‘Landsat Satellite’, and a subset of the ‘MNIST’ handwritten digits data set. Experiments on various data sets can help answer the question if the selection of the AEs in conjunction with SOMs is data-dependent or not. The performance evaluation methods of the SOM are based on the quality measures proposed by L. Hamel [9], which is based on map embedding accuracy and estimated topographic accuracy.

The remaining chapters of this thesis are organized as follows:

Chapter 2 Literature Review: introduce the theory and relevant research of SOMs, variations of AEs, and Deep Neural Map based on various literature.

Chapter 3 Methodology: explain the experiment design, dataset selection, evaluation methods, and model implementation details.

Chapter 4 Results: show the experiment results, compare and evaluate the performance of each model.

Chapter 5 Conclusion: summarize the results I obtained, propose for future work.

CHAPTER 2

Literature Review

2.1 Self-Organizing Map

A kind of artificial neural network created by Teuvo Kohonen [2], the Self-Organizing Map (SOM), is an unsupervised learning algorithm that is mainly used for the visualization of high-dimensional data. Usually, it produces a two-dimensional lattice of nodes (called a map) to represent the high-dimensional input data while preserving the topological relationships of the input [2], and therefore it is utilized in dimensionality reduction. The convergence of the SOM algorithm has been proved by Y. Cheng [10], the model will converge after reasonably long iterations [2].

The basic SOM algorithm can be summarized as follows [11]:

1) Selective step: initialize each node's weight vectors randomly, select a training data vector $\mathbf{x}_k$ from the input space.

2) Competitive step: find the best matching neuron based on the Euclidean distance between the data vector $\mathbf{x}_k$ and the neurons:

$$c = \operatorname{argmin}_i(\|\mathbf{m}_i - \mathbf{x}_k\|) \tag{1}$$

where m_i is a neuron indexed by i and c denotes the index of the best matching neuron $\mathbf{m}_c$ on the map.

3) Update step: update the winning neuron's neighborhood using the following rule:

$$\mathbf{m}_i \leftarrow \mathbf{m}_i - \eta(\mathbf{m}_i - \mathbf{x}_k)h(c, i) \tag{2}$$

where $\eta(\mathbf{m}_i - \mathbf{x}_k)$ denotes the difference between the neuron and the training instance scaled by the learning rate η $(0 < \eta < 1)$, $h(c, i)$ denotes the following loss function:

$$h(c, i) = \begin{cases} 1 & if\ i \in \Gamma(c), \\ 0 & otherwise, \end{cases} \tag{3}$$

where $\Gamma(c)$ is the neighborhood of the best matching neuron m_c.

Repeat from the selective step for N iterations until the model converges. For a large high-dimensional data set, N could be a large number, however, the basic SOM does not show a high performance after reasonably long iterations [11].

2.1.1 Vectorized SOM Training

Vectorized SOM training (VSOM) proposed by L. Hamel [11] is an efficient implementation of stochastic training for SOMs, which replaces all iterative constructs with vector and matrix operations. It is a single threaded algorithm, providing substantial performance increases over the basic SOM algorithm (up to 60 times faster)[11]. Because R does not support multi-threading well, the VSOM is well suited as a replacement for iterative stochastic training of SOM in R [11]. The VSOM implementation is available in R based POPSOM package [12].

2.2 Autoencoder

The origin of the autoencoder (AE) is not clear and the terminology may change over time. J. Schmudhuber [13] indicates that perhaps the first work to study potential benefits of unsupervised learning based pre-training was published by Dana H. Ballard [14] in 1987, which proposed unsupervised AE hierarchies. According to the information provided in [15], I summarize the AE as following.

An AE is a kind of artificial neural network that is mainly used for feature extraction and dimensionality reduction. It is composed of two parts, an encoder and a decoder, which aims to reconstruct the original input. The encoder maps the input into a hidden layer representation (or called code), and then the decoder reconstructs the input from the hidden layer representation.

An autoencoder could be undercomplete or overcomplete. The one with code dimension less than the input dimension is called undercomplete, while the one with code dimension greater than the input dimension is called overcomplete. Regularization can prevent the overcomplete autoencoder from only copying the input to the output without learning anything useful [15], such as sparse autoencoder, denoising autoencoder, and contractive autoencoder.

2.2.1 Sparse Autoencoder

In 1997, Olshausen and Field [16] indicated that sparse coding with an overcomplete basis set leads to interesting interactions among the code elements because sparsification weeds out those basis functions not needed to describe a given image structure. Hence, sparse coding is a good candidate for the data set whose input data contain much noise [17].

Sparse autoencoder (SAE) is a kind of overcomplete autoencoder that includes more hidden nodes than input, but only a small number of hidden nodes are activated at once [18]. The training criterion of an SAE involves a sparsity penalty $\Omega(\boldsymbol{h})$ on the code layer $\boldsymbol{h}$, in addition to the reconstruction error L, the objective function is as following [15]:

$$L\left(x, g\big(f(x)\big)\right) + \Omega(\boldsymbol{h}) \tag{4}$$

where $f(x)$ denotes the encoder output, $g(\boldsymbol{h})$ denotes the decoder output, we have $\boldsymbol{h} = (h_1, h_2, \dots, h_n) = f(x)$. The sparsity penalty $\Omega(\boldsymbol{h})$ can be formulated in different ways, and one approach is applying L1 regularization term on the activation and scaling by a tuning hyperparameter λ [15]:

$$\Omega(\boldsymbol{h}) = \lambda \sum_i |h_i| \tag{5}$$

Recently, an autoencoder with linear activation function called K-Sparse Autoencoder [19] was proposed in 2013, in which only the k-highest activities are kept in hidden layers. It achieves high speed on the encoding stage and well-suits to large problem sizes[19].

2.2.2 Denoising Autoencoder

Differently from SAE that adds a penalty to the loss function, the denoising autoencoder (DAE) achieves a representation by changing the reconstruction error term of the loss function [15]. The DAE takes corrupted input data and is trained to predict the original uncorrupted data as output [15], therefore the input and output for a DAE are no longer the same. Figure 2 shows the architecture of a DAE:

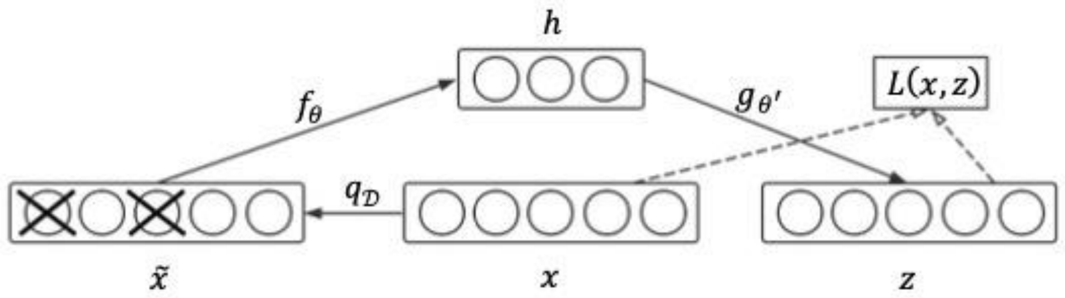


Figure 2. The DAE architecture. Reproduced from ref [20]

the initial input x is corrupted into $\tilde{x}$ by stochastic mapping $\tilde{x} \sim q_{\mathcal{D}}(\tilde{x}|x)$, the encoder then maps it to a hidden representation $h = f_{\theta}(\tilde{x})$ from which we reconstruct the $z = g_{\theta'}(h)$, and the reconstruction error is measured by loss $L(x, z)$ [21]. In order to let reconstruction z as close as possible to the clean input x, the parameters θ and θ' are trained to minimize the average reconstruction error over the training set [21]. Note that the corruption process $q_{\mathcal{D}}(\tilde{x}|x)$ could be any types, such as Gaussian noise, Masking noise, and Salt-and-pepper noise [21].

2.2.3 Contractive Autoencoder

The contractive autoencoder (CAE) aims to resist perturbations of the input and is encouraged to contract the input neighborhood to a smaller output neighborhood [15]. CAE adds a regularizer penalty $\|J_f(x)\|_F^2$ (the Frobenius norm of the Jacobian matrix $J_f(x)$) to the reconstruction cost function to encourage robustness of the representation $f(x)$ [22]:

$$\|J_f(x)\|_F^2 = \sum_{ij} \left(\frac{\partial h_j(x)}{\partial x_i} \right)^2 \tag{6}$$

where h is the hidden representation, the penalty is the sum of squares of all partial derivatives of the extracted features $h(x)$ with respect to the input x [22]. Similar as SAE, the objective function of the CAE has the following form:

$$\mathrm{L}\left(x, g\big(f(x)\big)\right) + \lambda \|J_f(x)\|_F^2 \tag{7}$$

By comparing CAEs with DAEs, we can see that CAEs encourage robustness of representation $f(x)$, but DAEs encourage robustness of reconstruction $g(f(x))$ [22]. In 2014, Alain and Bengio [23] showed that in the limit of small Gaussian input noise

DAEs make the reconstruction function resistant to finite-sized perturbations of the input, but CAEs make the reconstruction function resistant to infinitesimal perturbations of the input [15].

2.2.4 Convolutional Autoencoder

Different from basic autoencoders, a convolutional autoencoder (ConvAE) is built with convolutional layers rather than fully connected layers, hence it is efficient for image data sets. To exploit the spatial structure of images, the convolutional autoencoder is defined as follow [24]:

$$f_W(x) = \sigma(x * W) = h$$
$$g_U(h) = \sigma(h * U) \tag{8}$$

where $f_W(x)$ denotes the encoder output, $g_U(h)$ denotes the decoder output, x and the embedded code h are matrices or tensors, σ is the activation function, and $*$ is convolution operator. The object is to minimize the mean squared errors between the input and output over all samples [24]:

$$\min_{W,U} \frac{1}{n} \sum_{i=1}^{n} \left\| g_U\left(f_W(x_i)\right) - x_i \right\|_2^2 \tag{9}$$

In recent research, a Fully Convolutional Autoencoder (FCAE) [25] was proposed in 2017 which can be trained in an end-to-end manner. It is composed of convolution (de-convolution) layers and pooling (un-pooling) layers, plus adding batch normalization layers to each of the convolution-type layers. Different from the traditional ConvAEs, the FCAE could avoid the tedious and time-consuming layer-wise pretraining stage [25].

2.3 Deep Neural Map

A new Deep Neural Maps (DNMs) model designed by Mehran Pesteie, Purang Abolmaesumi and Robert R. Rohling [4] in 2018 gives excellent performance in high-dimensional data visualization, which uses SOM models in conjunction with deep convolutional AEs shown in Figure 3. The result shows that the DNM has separated each class of input data and mapped it to a particular position on a lattice successfully [4]. D. Rajashekar [3] proposed an Autoencoder based Self Organizing Map (AESOM) framework, which uses an AE that contains two hidden layers. It shows improvements in data representation and improves detection rates from encoding and reduces the feature space of the input [3].

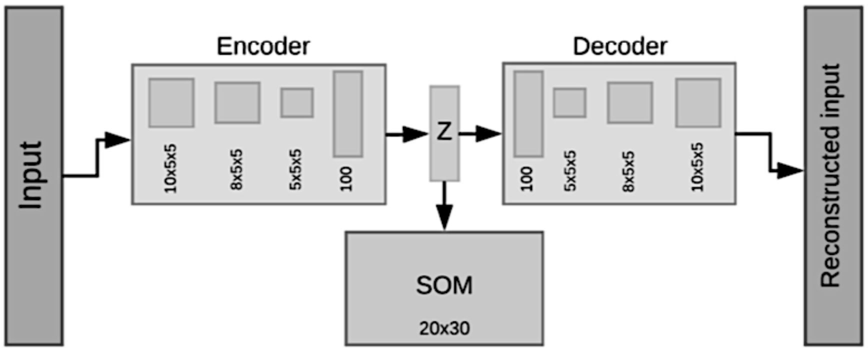


Figure 3. The DNM architecture. Reproduced from ref [4]

CHAPTER 3

Methodology

3.1 Experiment Design

In this research, the experiment is mainly divided into two parts: 1) implement the AEs and SOMs (build five AEs with Keras[26] in TensorFlow[27] library and implement SOMs with the R-based POPSOM library[12]), 2) evaluate the performance. In this chapter, I will introduce the evaluation methods and implementation process in detail.

3.1.1 Model Structure

Based on the DNM model, the overall model structure is shown in Figure 4. First, I input the original data to each of the five types of AEs (basic AE, SAE, DAE, CAE, ConvAE), then extract the encoded data (embedding) and input it to SOMs. I also input the original data to SOMs as a contrast experiment. Moreover, I measure the reconstruction error between the input and the reconstructed input and evaluate the quality of SOMs.

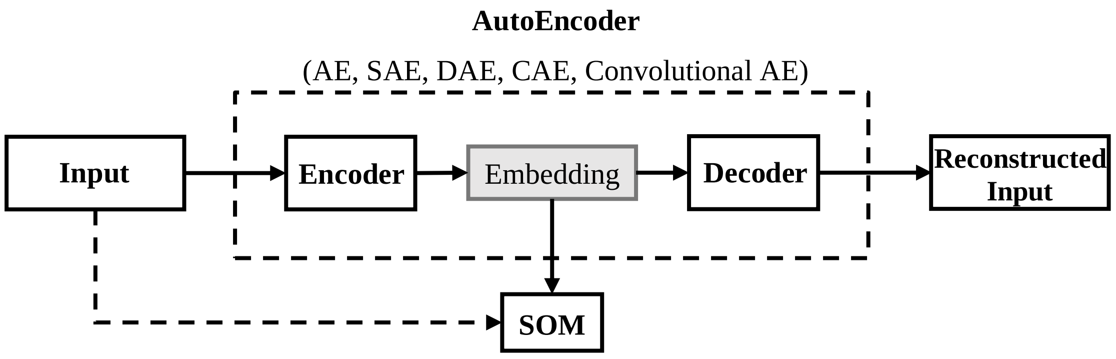
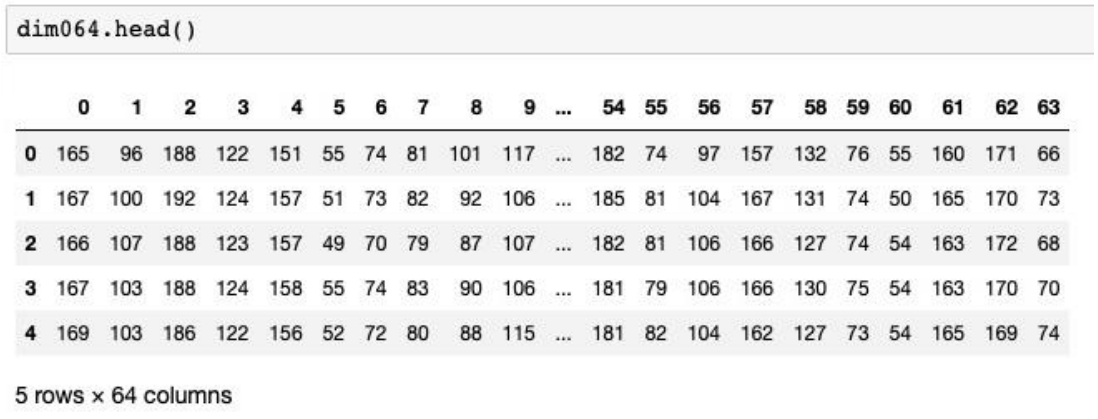


Figure 4. A Variant of DNM model architecture

3.1.2 Data Set Selection

In this project, the task of AEs is reducing dimensionality and extracting features, and the task of SOMs is clustering the input data. For this purpose, the ideal data set for this project is the one with high dimensionality and precise classification. To compare and to evaluate the performance of AEs in conjunction with SOMs in various circumstances, three different types (synthetic, real-world, image) of data sets were selected.

1) The ‘dim064’ [28] [29] is a 64-dimensional synthetic data set with 1024 observations that well separated in 16 Gaussian clusters (Figure 5). I split the data set with a ratio of 0.4, namely 60% data for training (614 instances) and 40% data for testing (410 instances).



```
dim064.head()
```

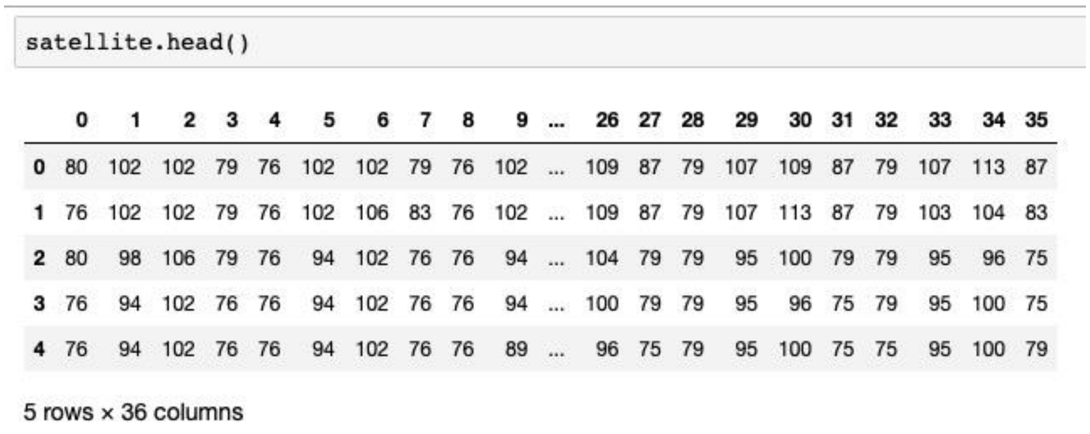
	0	1	2	3	4	5	6	7	8	9	...	54	55	56	57	58	59	60	61	62	63
0	165	96	188	122	151	55	74	81	101	117	...	182	74	97	157	132	76	55	160	171	66
1	167	100	192	124	157	51	73	82	92	106	...	185	81	104	167	131	74	50	165	170	73
2	166	107	188	123	157	49	70	79	87	107	...	182	81	106	166	127	74	54	163	172	68
3	167	103	188	124	158	55	74	83	90	106	...	181	79	106	166	130	75	54	163	170	70
4	169	103	186	122	156	52	72	80	88	115	...	181	82	104	162	127	73	54	165	169	74

5 rows × 64 columns

Figure 5. The head five rows of ‘dim064’ data set

2) The ‘Landsat Satellite’ from UCI machine learning repository [30] is a real-world data set with 6435 instances and 36 attributes that categorized in 6 classes (Figure 6). The data set consists of the multi-spectral values of pixels in 3 by 3 neighborhoods in a satellite image, and the classification associated with the central

pixel in each neighborhood. The original image cannot be reconstructed because the data is given in random order and the certain lines of data have been removed. The 36 attributes (4 spectral bands multiply by 9 pixels in neighborhood) are numerical in the range 0 to 255, the 6 classes of pixels are coded as numbers (1: red soil, 2: cotton crop, 3: grey soil, 4: damp grey soil, 5: soil with vegetation stubble, and 7: very damp grey soil). The training set contains 4435 instances, and the test set contains 2000 instances.



```
satellite.head()
```

	0	1	2	3	4	5	6	7	8	9	...	26	27	28	29	30	31	32	33	34	35
0	80	102	102	79	76	102	102	79	76	102	...	109	87	79	107	109	87	79	107	113	87
1	76	102	102	79	76	102	106	83	76	102	...	109	87	79	107	113	87	79	103	104	83
2	80	98	106	79	76	94	102	76	76	94	...	104	79	79	95	100	79	79	95	96	75
3	76	94	102	76	76	94	102	76	76	94	...	100	79	79	95	96	75	79	95	100	75
4	76	94	102	76	76	94	102	76	76	89	...	96	75	79	95	100	75	75	95	100	79

5 rows × 36 columns

Figure 6. The head five rows of ‘Landsat Satellite’ data set

3) The ‘MNIST’ database [31]is a large database of handwritten digits that is widely used for machine learning. It consists of 70,000 (60,000 for training, 10,000 for testing) grey-scale images of handwritten digits (‘0’ – ‘9’) whose size is 28 by 28 pixels. I selected 10,000 examples from the training set and 2,000 examples from the test set to make a subset of MNIST database that as my third data set.

I converted the original image into 28 by 28 2D-array and scaled the value of each cell between 0 and 1, and each cell represents the single pixel of the image. Before feeding to the AEs (except ConvAEs), the 2D array was flattened into a 1D array, hence the dimension of the data set is 784 (28 by 28).

3.2 Evaluation Methods

3.2.1 Performance Evaluation of AEs

The evaluation process of AEs is based on the loss error (reconstruction error). I plot the loss functions of training data and validation data for each type of AEs and compare the mean and minimum value of them. For image data sets, I also plot the original input images and the decoded images to show visible reconstruction results. Additionally, the evaluation results of SOMs also indicate the quality of AEs that whether the encoders extract useful features.

3.2.2 Performance Evaluation of SOMs

Within this research, the evaluation methods of SOMs are based on the SOM quality measures presented by L. Hamel [9], which is an efficient statistical approach measures both the embedding and the topological quality of a SOM.

1) Embedding Accuracy

The motivation for the map embedding accuracy is that [9], ‘A SOM is completely embedded if its neurons appear to be drawn from the same distribution as the training instances.’ That features are embedded means that their mean and variance are adequately modeled by the neurons in the SOM. The embedding accuracy (ea) for d features are defined as following:

$$ea = \frac{1}{d}\sum_{i=1}^{d} \rho_i\, , \tag{10}$$

where

$$\rho_i = \begin{cases} 1 & \text{if feature } i \text{ is embedded,} \\ 0 & \text{otherwise.} \end{cases} \tag{11}$$

A map is fully embedded if the embedding accuracy equals 1.

2) Estimated Topographic Accuracy

The topographic error [32] is almost the simplest measure of the topological quality of a map which is defined as:

$$te = \frac{1}{n}\sum_{i=1}^{n} err(x_i) \tag{12}$$

where

$$err(x_i) = \begin{cases} 1 & \text{if } bmu(x_i) \text{ and } 2bmu(x_i) \text{are not neighbors,} \\ 0 & \text{otherwise.} \end{cases}$$

n is the number of training instances, x_i denotes the ith training vector on the map, $bmu(x_i)$ and $2bmu(x_i)$ are the best matching unit and the second-best matching unit for x_i. The estimated topographic accuracy [9] can be defined as,

$$ta' = 1 - \frac{1}{s}\sum_{i=1}^{s} err(x_i) \tag{13}$$

where s is the size of a sample S of the training data. L. Hamel indicated that we can get accurate values for ta' with very small samples so that the algorithm is more efficient than conventional topographic accuracy (1 - te). We say a map is fully organized if the topographic accuracy close to 1.

3) Convergence Accuracy

Convergence accuracy is an SOM quality assessment which is implemented in the R-based POPSOM package [12] [33]. It is defined as,

$$ca = \frac{1}{2}\, ea + \frac{1}{2}\, ta' \tag{14}$$

The convergence accuracy is a linear combination of the embedding accuracy and the estimated topographic accuracy, which indicates the model performance from both the training data set and the map neurons. It is the primary approach to evaluate and compare the quality of SOMs in this research.

3.3 Implementation

The five types of AEs were implemented in Python with the TensorFlow Keras framework. The SOMs were built in R with the POPSOM package.

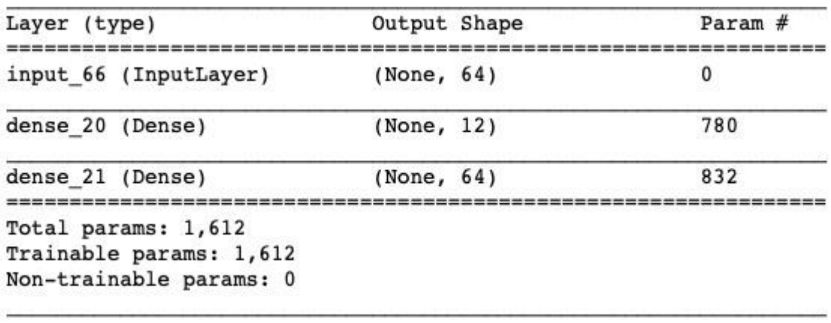
3.3.1 Basic AE

I implemented a single fully-connected layer as encoder and as decoder. The parameters of the basic AE for each data set are shown in Table 1, Table 2, and Table 3. The architecture of the basic AE for each data set are shown in Figure 7, Figure 8, and Figure 9.

1) dim064 data set

Table 1. Parameters of basic AE in ‘dim064’

Encoding Dimensionality	12
Encoder Activation	relu
Decoder Activation	sigmoid
Optimizer	adam
Loss	mean squared error



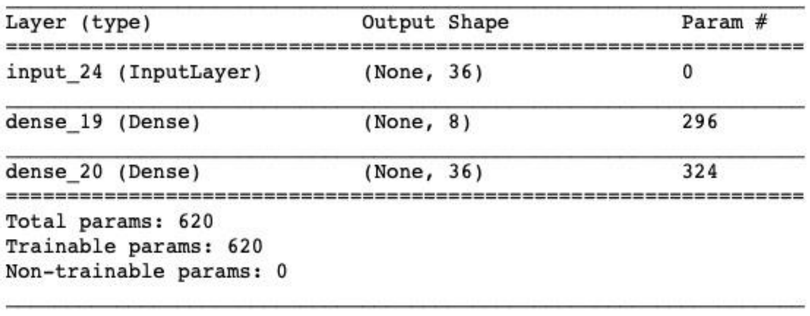
```
Layer (type)                 Output Shape              Param #
=================================================================
input_66 (InputLayer)        (None, 64)                0
_________________________________________________________________
dense_20 (Dense)             (None, 12)                780
_________________________________________________________________
dense_21 (Dense)             (None, 64)                832
=================================================================
Total params: 1,612
Trainable params: 1,612
Non-trainable params: 0
_________________________________________________________________
```

Figure 7. Architecture of basic AE in ‘dim064’

2) Landsat Satellite data set

Table 2. Parameters of basic AE in ‘Landsat Satellite’

Encoding Dimensionality	8
Encoder Activation	relu
Decoder Activation	sigmoid
Optimizer	adam
Loss	mean squared error



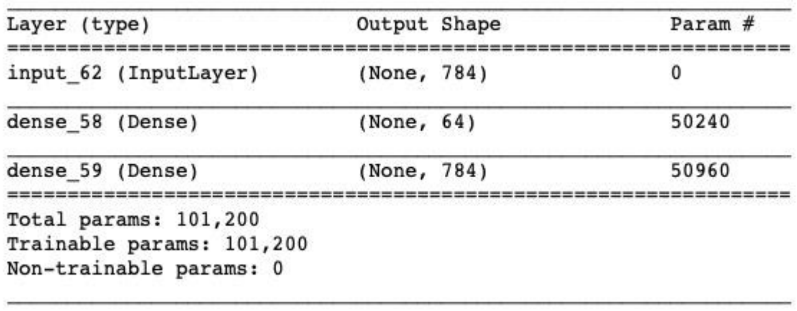
```
Layer (type)                 Output Shape              Param #
=================================================================
input_24 (InputLayer)        (None, 36)                0
_________________________________________________________________
dense_19 (Dense)             (None, 8)                 296
_________________________________________________________________
dense_20 (Dense)             (None, 36)                324
=================================================================
Total params: 620
Trainable params: 620
Non-trainable params: 0
_________________________________________________________________
```

Figure 8. Architecture of basic AE in ‘Landsat Satellite’

3) Subset of MNIST data set

Table 3. Parameters of basic AE in ‘MNIST’

Encoding Dimensionality	64
Encoder Activation	relu
Decoder Activation	sigmoid
Optimizer	adadelta
Loss	binary cross entropy



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_62 (InputLayer)        (None, 784)               0         
_________________________________________________________________
dense_58 (Dense)             (None, 64)                50240     
_________________________________________________________________
dense_59 (Dense)             (None, 784)               50960     
=================================================================
Total params: 101,200
Trainable params: 101,200
Non-trainable params: 0
_________________________________________________________________
```

Figure 9. Architecture of basic AE in ‘MNIST’

3.3.2 SAE

The SAE adds an L1 regularizer to the encoded layer base on the basic AE. Both the parameters (Table 1, Table 2, Table 3) and architecture (Figure 7, Figure 8, Figure 9) of SAE for each data set are the same as the basic AE.

3.3.3 CAE

The CAE uses the same parameters (Table 1, Table 2, Table 3) and architecture (Figure 7, Figure 8, Figure 9) as the basic AE as well, except that a different loss function is applied. According to the objective function (Equation 7) of the CAE, I implemented a distinct loss function by expanding the Equation 6 as,

$$\|J_f(x)\|_F^2 = \sum_{ij}\left(\frac{\partial h_j(x)}{\partial x_i}\right)^2$$

$$= \sum_j [f_j(1-f_j)]^2 \sum_i (W_{ji}^T)^2 \tag{15}$$

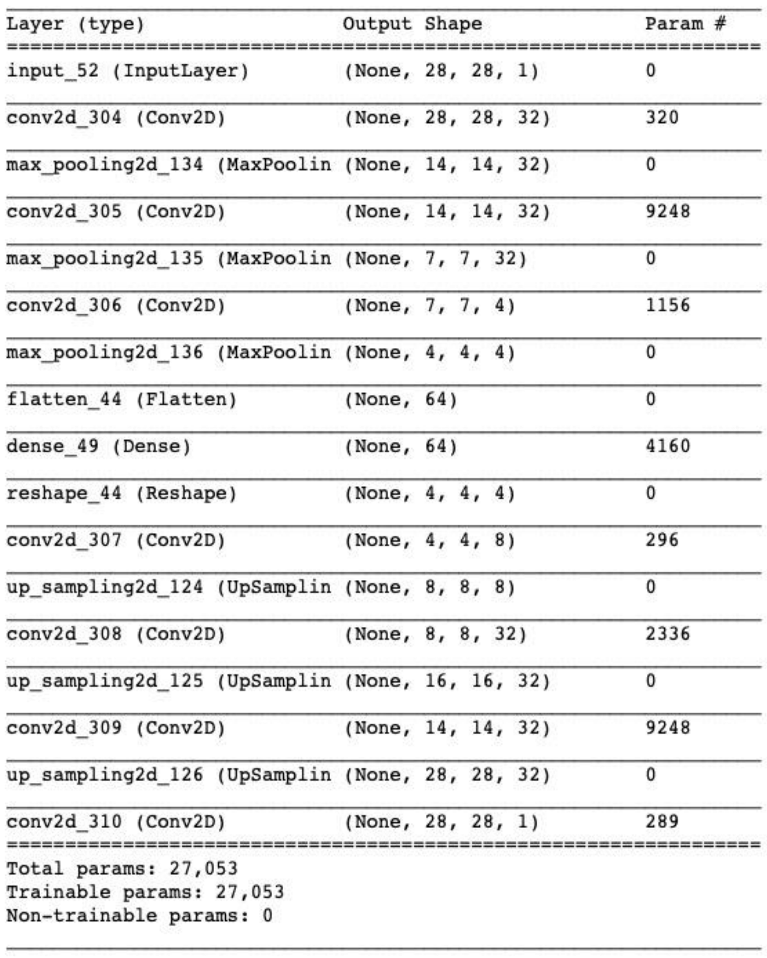
then translated the equation to Python code and got a contractive loss function [34].

3.3.4 DAE

I set the noise factor to be 0.5 to create noisy input. For ‘dim064’ and ‘Landsat Satellite’ data sets, both the encoded layer and the decoded layer are still single fully-connected layers, and the parameters are the same as before. For the subset of MNIST data set, I implemented a Denoising Convolutional Autoencoder (DCAE), the architecture is shown in Figure 10. Before feeding to the network, I reshaped each input into size $28 \times 28 \times 1$.

The encoder consists of three 2D convolutional layers followed by down-sampling (max-pooling) layers (pooling size 2×2) and a flatten layer (encoded layer). The first two convolutional layers have 32 filters and the third one has 4 filters of size 3×3. The output of the encoded layer is 64 dimensional.

The decoder consists of four 2D convolutional layers followed by three up-sampling layers (size 2×2), the last convolutional layer is the decoded layer. The first convolutional layer has 8 filters, the following two convolutional layers have 32 filters, and the decoded layer has 1 filter of size 3×3.

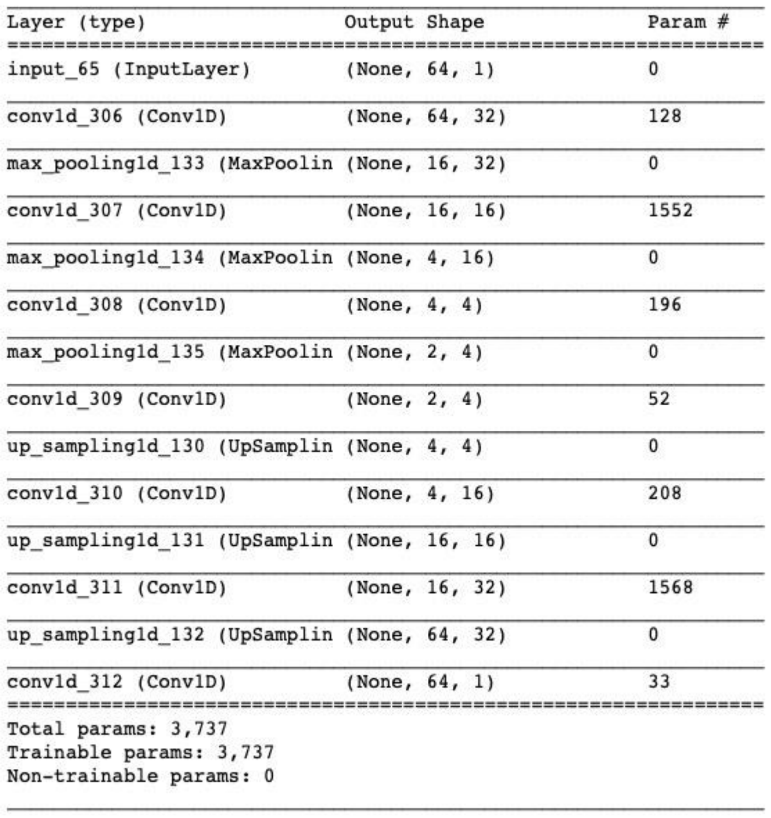


```
Layer (type)                 Output Shape              Param #
=================================================================
input_52 (InputLayer)        (None, 28, 28, 1)         0
_________________________________________________________________
conv2d_304 (Conv2D)          (None, 28, 28, 32)        320
_________________________________________________________________
max_pooling2d_134 (MaxPoolin (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_305 (Conv2D)          (None, 14, 14, 32)        9248
_________________________________________________________________
max_pooling2d_135 (MaxPoolin (None, 7, 7, 32)          0
_________________________________________________________________
conv2d_306 (Conv2D)          (None, 7, 7, 4)           1156
_________________________________________________________________
max_pooling2d_136 (MaxPoolin (None, 4, 4, 4)           0
_________________________________________________________________
flatten_44 (Flatten)         (None, 64)                0
_________________________________________________________________
dense_49 (Dense)             (None, 64)                4160
_________________________________________________________________
reshape_44 (Reshape)         (None, 4, 4, 4)           0
_________________________________________________________________
conv2d_307 (Conv2D)          (None, 4, 4, 8)           296
_________________________________________________________________
up_sampling2d_124 (UpSamplin (None, 8, 8, 8)           0
_________________________________________________________________
conv2d_308 (Conv2D)          (None, 8, 8, 32)          2336
_________________________________________________________________
up_sampling2d_125 (UpSamplin (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_309 (Conv2D)          (None, 14, 14, 32)        9248
_________________________________________________________________
up_sampling2d_126 (UpSamplin (None, 28, 28, 32)        0
_________________________________________________________________
conv2d_310 (Conv2D)          (None, 28, 28, 1)         289
=================================================================
Total params: 27,053
Trainable params: 27,053
Non-trainable params: 0
_________________________________________________________________
```

Figure 10. Architecture of DCAE in ‘MNIST’

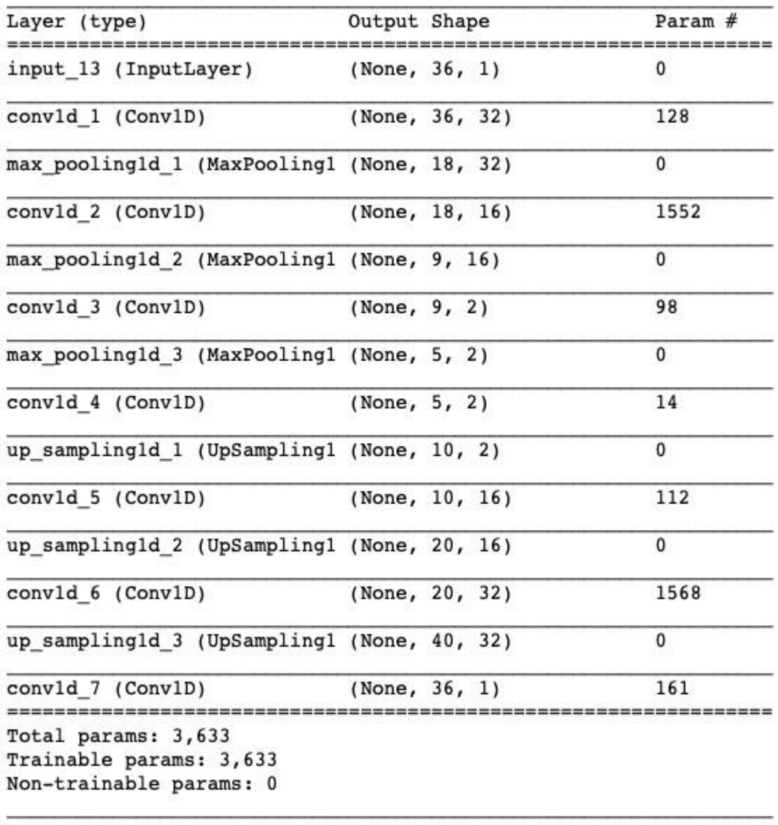
3.3.4 ConvAE

For the ‘dim064’ and the ‘Landsat Satellite’ data sets, I utilized 1D convolutional layers, 1D max-pooling layers, and 1D up-sampling layers to build the models. The architectures of convolutional AEs for these two data sets are shown in Figure 11 and Figure 12. Both architectures consist of the same types of neural network layers and are adapted to the input shapes of the data, which causes some differences in intermediate layers between the two.



```
Layer (type)                 Output Shape              Param #
=================================================================
input_65 (InputLayer)        (None, 64, 1)             0
_________________________________________________________________
conv1d_306 (Conv1D)          (None, 64, 32)            128
_________________________________________________________________
max_pooling1d_133 (MaxPoolin (None, 16, 32)            0
_________________________________________________________________
conv1d_307 (Conv1D)          (None, 16, 16)            1552
_________________________________________________________________
max_pooling1d_134 (MaxPoolin (None, 4, 16)             0
_________________________________________________________________
conv1d_308 (Conv1D)          (None, 4, 4)              196
_________________________________________________________________
max_pooling1d_135 (MaxPoolin (None, 2, 4)              0
_________________________________________________________________
conv1d_309 (Conv1D)          (None, 2, 4)              52
_________________________________________________________________
up_sampling1d_130 (UpSamplin (None, 4, 4)              0
_________________________________________________________________
conv1d_310 (Conv1D)          (None, 4, 16)             208
_________________________________________________________________
up_sampling1d_131 (UpSamplin (None, 16, 16)            0
_________________________________________________________________
conv1d_311 (Conv1D)          (None, 16, 32)            1568
_________________________________________________________________
up_sampling1d_132 (UpSamplin (None, 64, 32)            0
_________________________________________________________________
conv1d_312 (Conv1D)          (None, 64, 1)             33
=================================================================
Total params: 3,737
Trainable params: 3,737
Non-trainable params: 0
_________________________________________________________________
```

Figure 11. Architecture of ConvAE in ‘dim064’



```
Layer (type)                 Output Shape              Param #
=================================================================
input_13 (InputLayer)        (None, 36, 1)             0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 36, 32)            128
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 18, 32)            0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 18, 16)            1552
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 9, 16)             0
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 9, 2)              98
_________________________________________________________________
max_pooling1d_3 (MaxPooling1 (None, 5, 2)              0
_________________________________________________________________
conv1d_4 (Conv1D)            (None, 5, 2)              14
_________________________________________________________________
up_sampling1d_1 (UpSampling1 (None, 10, 2)             0
_________________________________________________________________
conv1d_5 (Conv1D)            (None, 10, 16)            112
_________________________________________________________________
up_sampling1d_2 (UpSampling1 (None, 20, 16)            0
_________________________________________________________________
conv1d_6 (Conv1D)            (None, 20, 32)            1568
_________________________________________________________________
up_sampling1d_3 (UpSampling1 (None, 40, 32)            0
_________________________________________________________________
conv1d_7 (Conv1D)            (None, 36, 1)             161
=================================================================
Total params: 3,633
Trainable params: 3,633
Non-trainable params: 0
_________________________________________________________________
```

Figure 12. Architecture of ConvAE in ‘Landsat Satellite’

For the subset of the MNIST data set, the architecture of ConvAE is the same as DCAE. Differently, input the original data to the network rather than the noisy data.

3.3.5 SOM

Before feeding the SOM with the encoded data extracted from five AEs, I drop the columns which are consisted of all zeros, because they contain no information for the clustering task. For the ‘dim064’ data set, I implemented a 20×15 map that has 300 neurons in total. For the ‘Landsat Satellite’ data set, I implemented a 40×35 map that has 1,400 neurons in total. For the subset of the ‘MNIST’ data set, I implemented a 40×40 map that has 1,600 neurons in total.

CHAPTER 4

Results

In this chapter, I will use the abbreviations shown in Table 4 to represent each model.

Table 4. Model abbreviation

Model Abbreviation	Input data of SOM encoded by
AE_SOM	Basic AE
SAE_SOM	SAE
CAE_SOM	CAE
DAE_SOM	DAE
DCAE_SOM	DCAE
ConvAE_SOM	ConvAE

4.1 'dim064' Experiment Results

4.1.1 Loss of AEs

After 200 epochs, the training loss and validation loss of each model are shown in Figure 13. All the models were trained well. For the DAE and ConvAE, the generalization of the models could not be further improved due to that the validation loss became saturated after approximately 150 epochs.

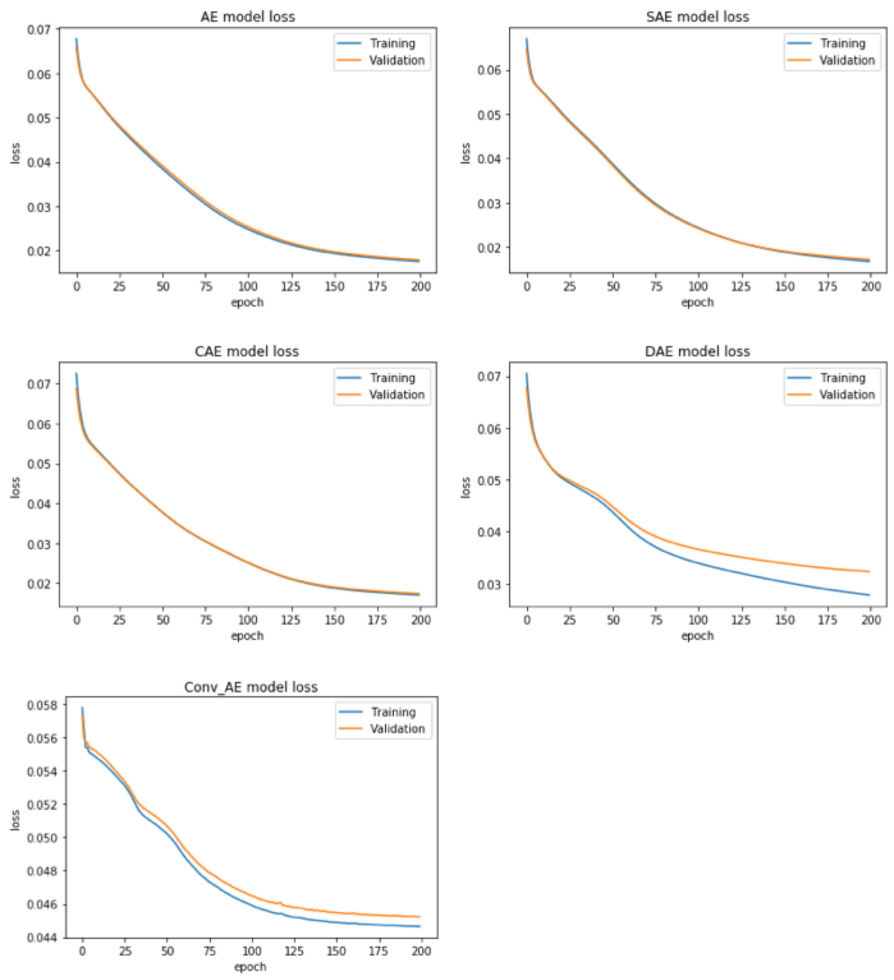


Figure 13. Training loss and validation loss of five models in ‘dim064’

4.1.2 SOM Models Results

I trained the SOM models from 10 to 400,000 (10, 100, 1000, 10,000, 50,000, 100,000, 200,000, 400,000) iterations for 5 times, plotted the convergence accuracy, embedding accuracy, and estimated topographic accuracy of each model, shown in Figure 14 and Figure 15. I scaled the x axis (iterations) as log base 2.

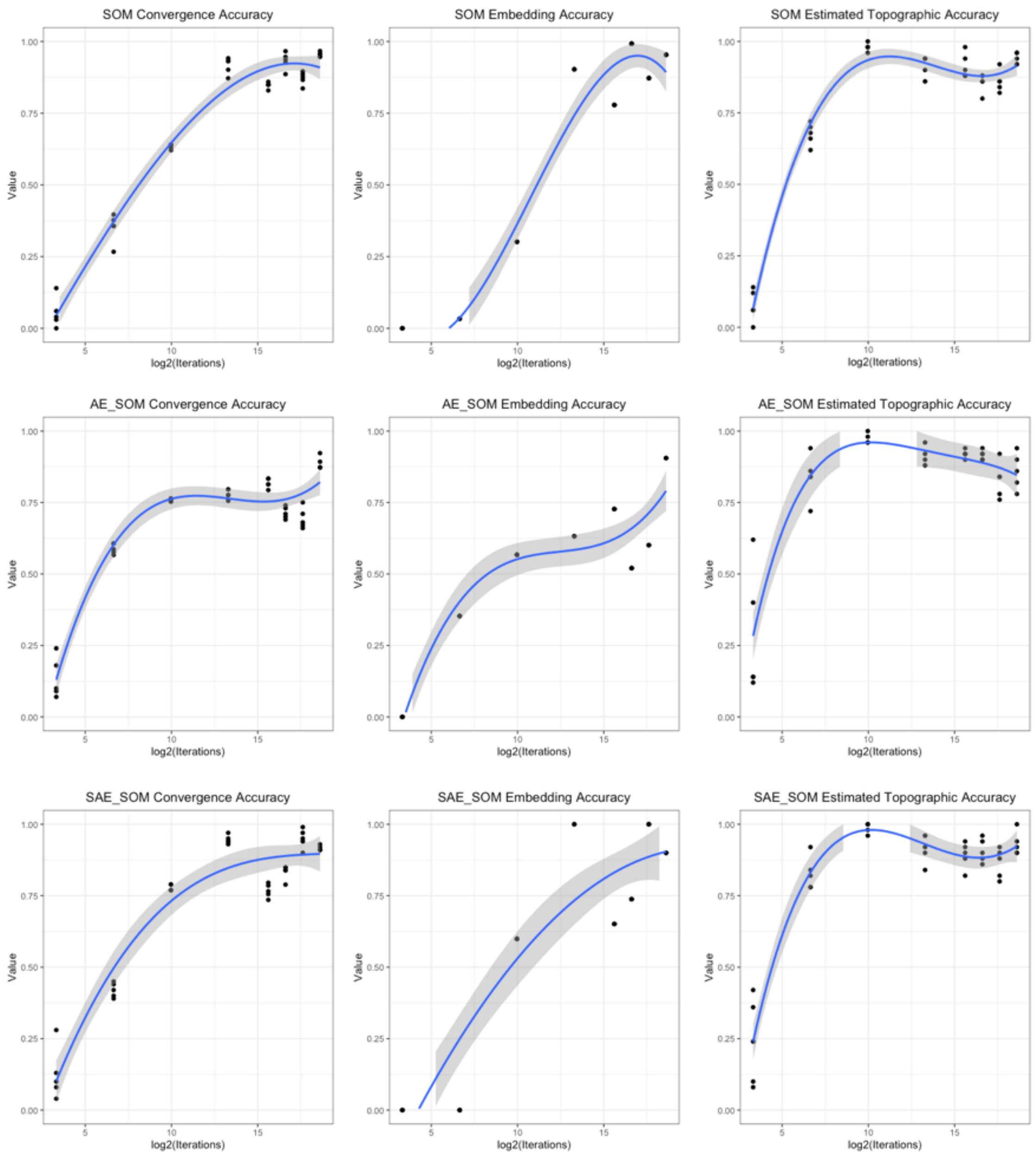


Figure 14. SOM, AE_SOM, SAE_SOM model quality measures in ‘dim064’

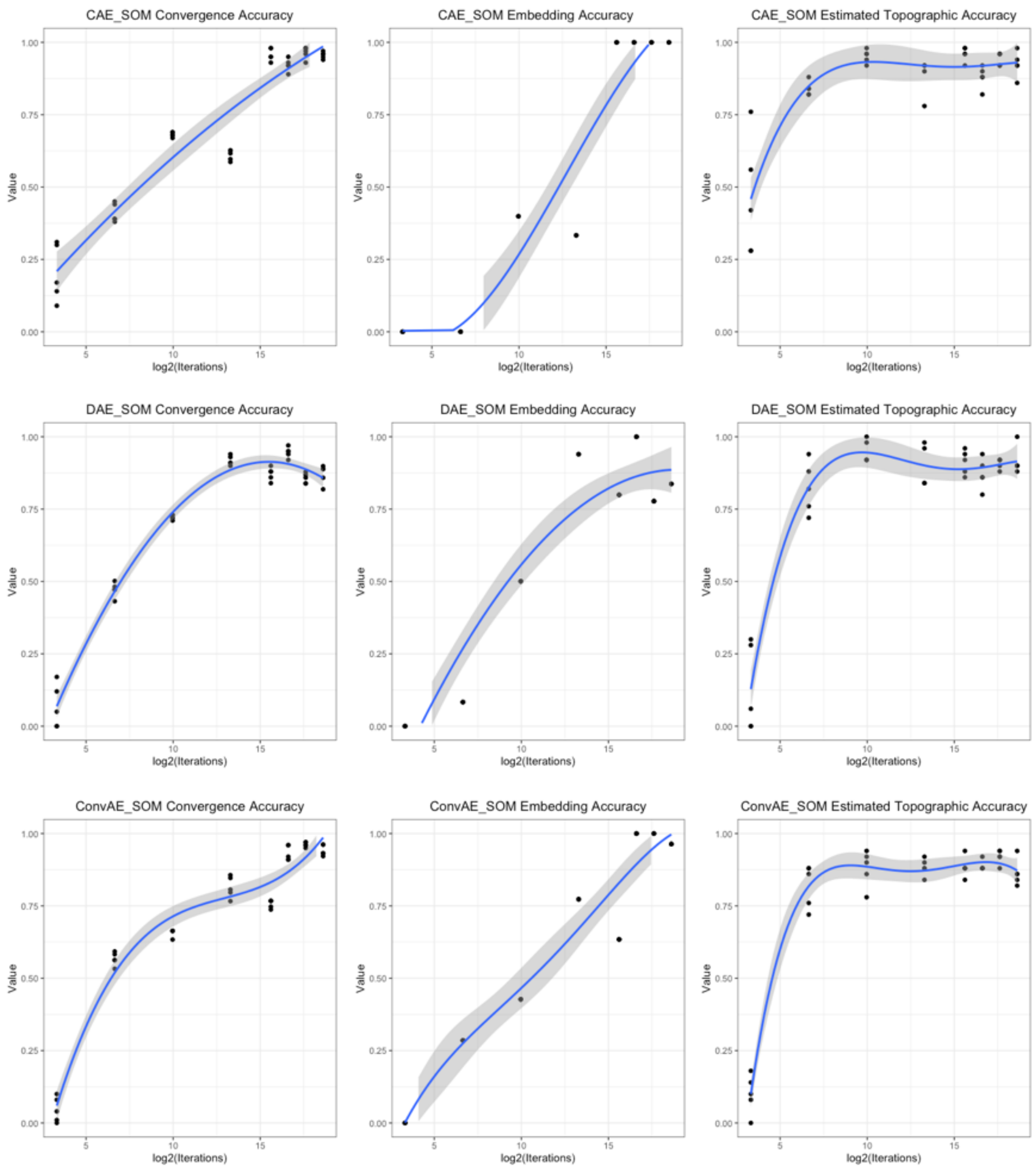


Figure 15. CAE_SOM, DAE_SOM, ConvAE_SOM model quality measures in 'dim064'

For the original data, the convergence accuracy varies around 0.88 after 10,000 iterations, both the embedding accuracy and the estimated topographic accuracy have oscillations after 10,000 iterations. The DAE_SOM shows very similar results as the SOM fed by original data.

For the AE_SOM, the convergence accuracy varies around 0.75, the embedding accuracy become oscillatory after 50,000 iterations, and the estimated topographic accuracy shows a downtrend, which indicates that the performance could not be better with more extended training. The embedding accuracy of SAE_SOM shows a similar trend as AE_SOM but with higher values, which up to 1, and the highest value of convergence accuracy is very close to 1. An appropriate iteration could help get better results for this model.

The CAE_SOM shows good results after 50,000 iterations. The embedding accuracy reaches the maximum 1, which shows that the neurons on the maps are perfectly drawn from the underlying distribution of training instances. The ConvAE_SOM also shows good embedding accuracy after 100,000 iterations, but the estimated topographic accuracy varies around 0.88 after 100 iterations and could not be further improved.

By comparison, CAE_SOM is the best, followed by ConvAE_SOM. This indicates that data have a property that they are insensitive to small perturbation so that CAE best captures their intrinsic structure. Except for the basic AE_SOM, using encoded data yields better results than using original data. Moreover, the encoding brings data to a lower dimensional representation, therefore it makes computing SOM more efficient.

Overall, all these models perform quite well in this dataset. The reason could be that synthetic datasets have a very good underlying clustering structure. Each feature in the data is equally important. The number of each category is averagely distributed among the data set. Little noise is persistent in the data. Thus, it is much easier for SOM to learn the actual distribution of training data even without encoding.

4.1.3 Clustering Result Representation

The starburst representation of the model (Figure 16) gives us a visible clustering result with class labels. The clusters are identified by light color (yellow) and cluster boundaries are identified by darker colors (red) [11]. The starburst lines help identify the center of each cluster, that all nodes are connected to their centroid node [33]. I plot the heat maps to confirm that those quantities (embedding accuracy, estimated topographic accuracy, convergence accuracy) when meeting certain criteria provide a good measure that SOM learns the underlying structure.

Since the CAE_SOM model achieved the best result, I implemented a 20×15 CAE_SOM compared with the SOM with unencoded data. I trained the models with 200,000 iterations and output the starburst representations of clusters, shown in Figure 16 and Figure 17.

Visibly, both maps separate the data into 14 clusters while two classes (with label 6 and label 7) are mis-clustered, and the locations of clusters distribute similarly on the maps. Overall, the clustering structure is almost the same, and CAE_SOM shows an excellent clustering result. Therefore, the encoded data has a similar structure to the original data, and both structures are successfully discovered by the SOM. It also suggests that we can trust the encoded data as the input of the SOM.

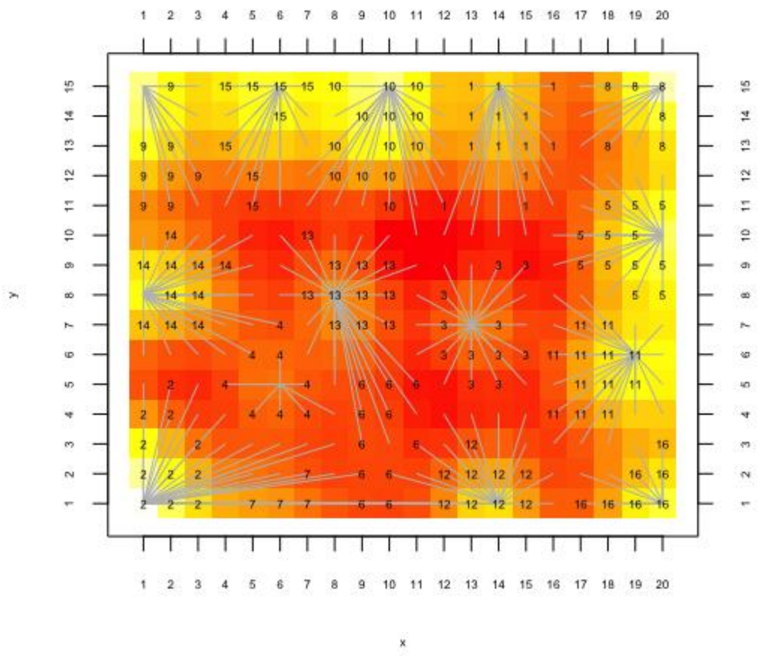


Figure 16. Starburst representation of CAE_SOM in 'dim064'

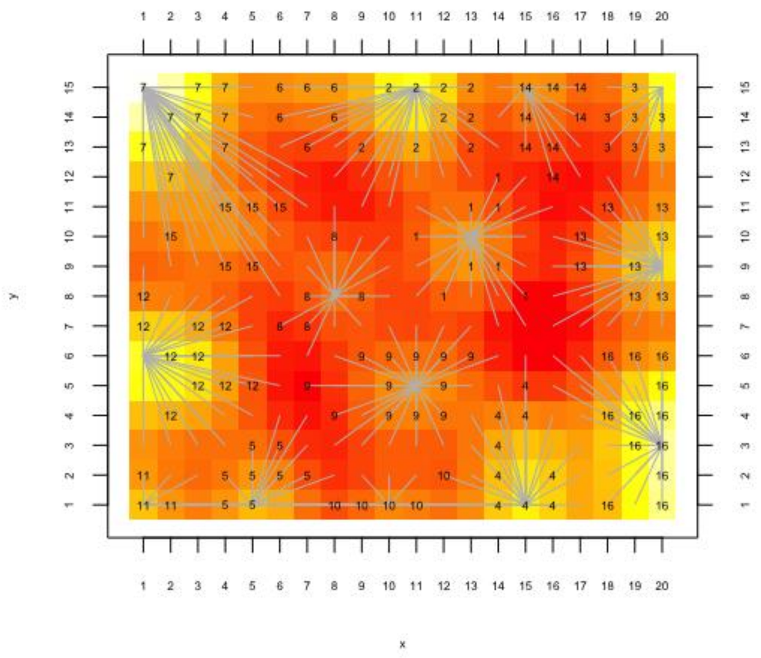


Figure 17. Starburst representation of SOM with unencoded data in 'dim064'

4.2 ‘Landsat Satellite’ Experiment Results

4.2.1 Loss of AEs

As seen from Figure 18, all the models were trained well after 200 epochs. For the DAE, the generalization of the models could not be further improved due to that the validation loss became saturated after approximately 175 epochs.

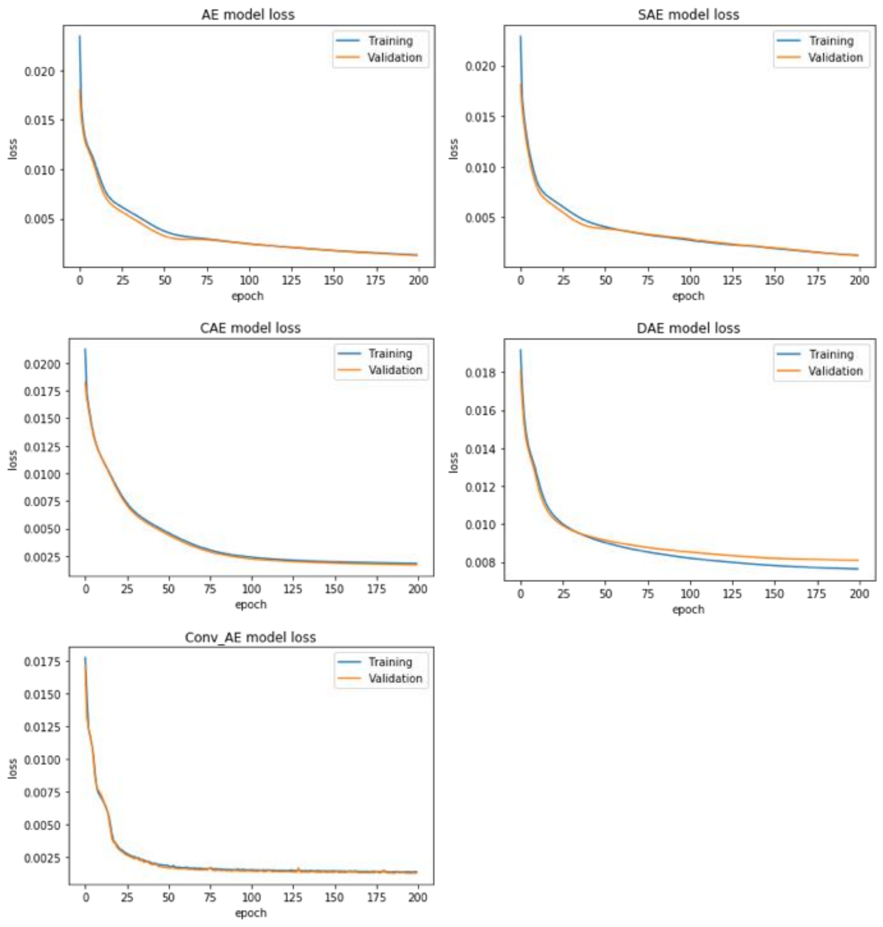


Figure 18. Training loss and validation loss of five models in ‘Landsat Satellite’

4.2.2 SOM Models Results

I trained the SOM models from 10 to 400,000 (10, 100, 1000, 10,000, 50,000, 100,000, 200,000, 300,000, 400,000) iterations for 5 times, plotted the convergence accuracy, embedding accuracy, and estimated topographic accuracy of each model, shown in Figure 19 and Figure 20. I scaled the x axis as log base 2.

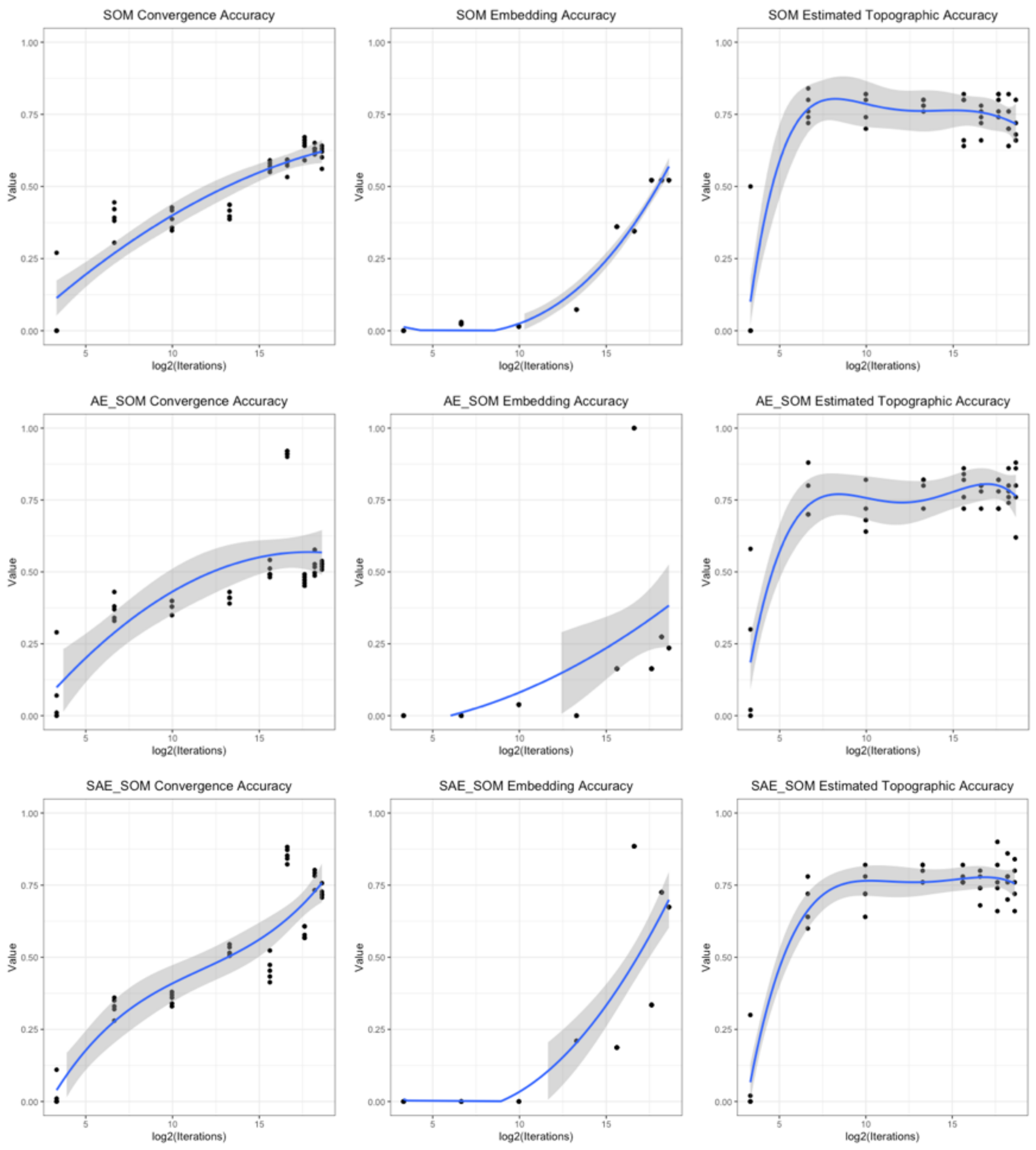


Figure 19. SOM, AE_SOM, SAE_SOM model quality measures in ‘Landsat Satellite’

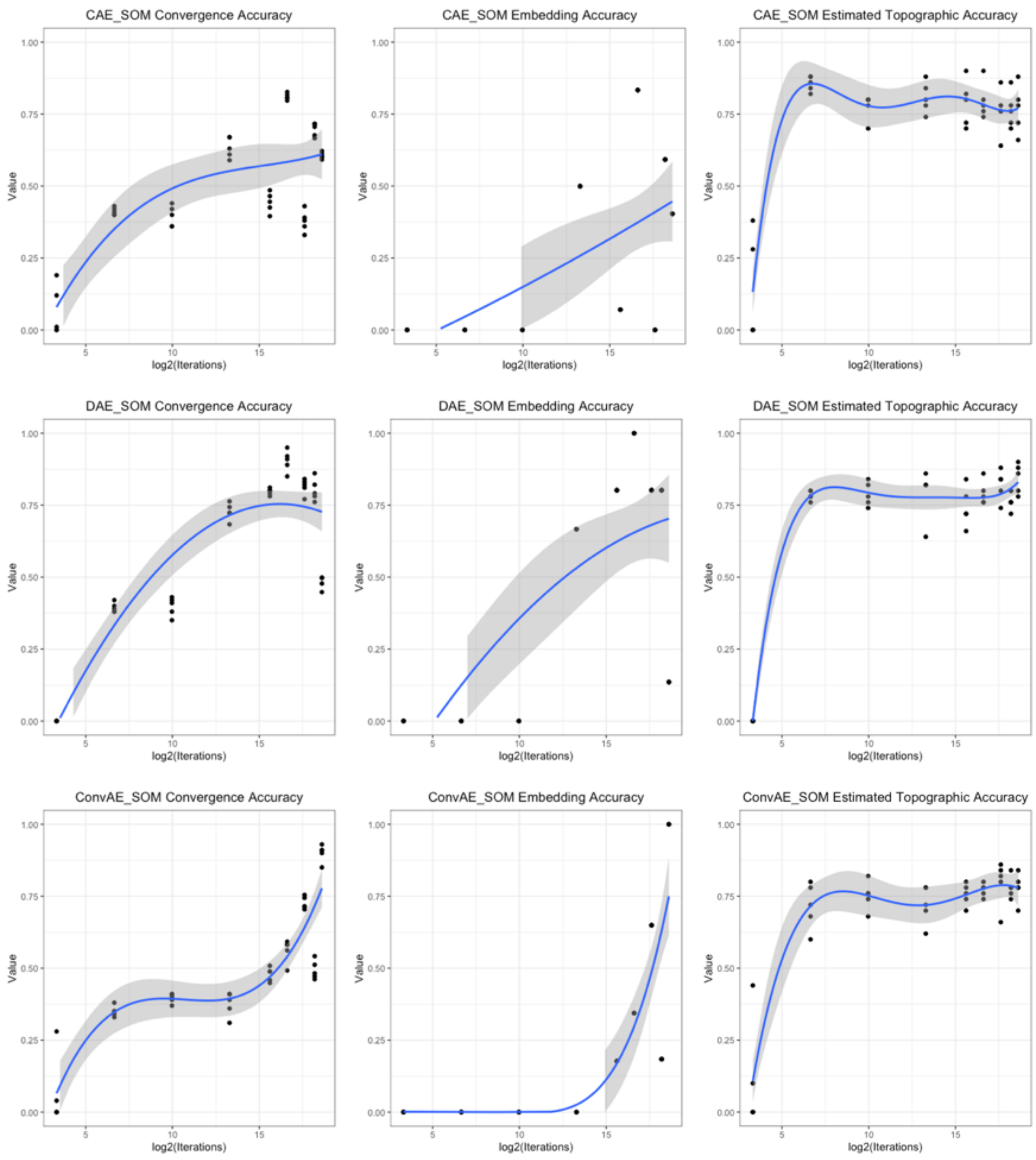


Figure 20. CAE_SOM, DAE_SOM, ConvAE_SOM model quality measures in 'Landsat Satellite'

All six models show similar results of estimate topographic accuracy, which varies around 0.75 and could not be further improved after roughly 1000 epochs. Moreover, they all show a consistently increasing trend in embedding accuracy with increasing training iterations. In rare cases, the model fed with encoded data could get a peak value of embedding accuracy at 100,000 iterations except for Conv_SOM.

Overall, the embedding accuracy of the AE_SOM model is below 0.5, which is the worst here. SAE_SOM and CAE_SOM only have a slightly better performance against AE_SOM. On the other hand, Conv_SOM and DAE_SOM have a more noticeable performance improvement after sufficient iterations. The reasons that Conv_SOM and DAE_SOM have a better performance could be due to that spectral data are extracted from images which contain observational noises. ConvAE is most suited to retrieve information in images, while DAE helps improve model robustness against noise.

4.2.3 Clustering Result Representation

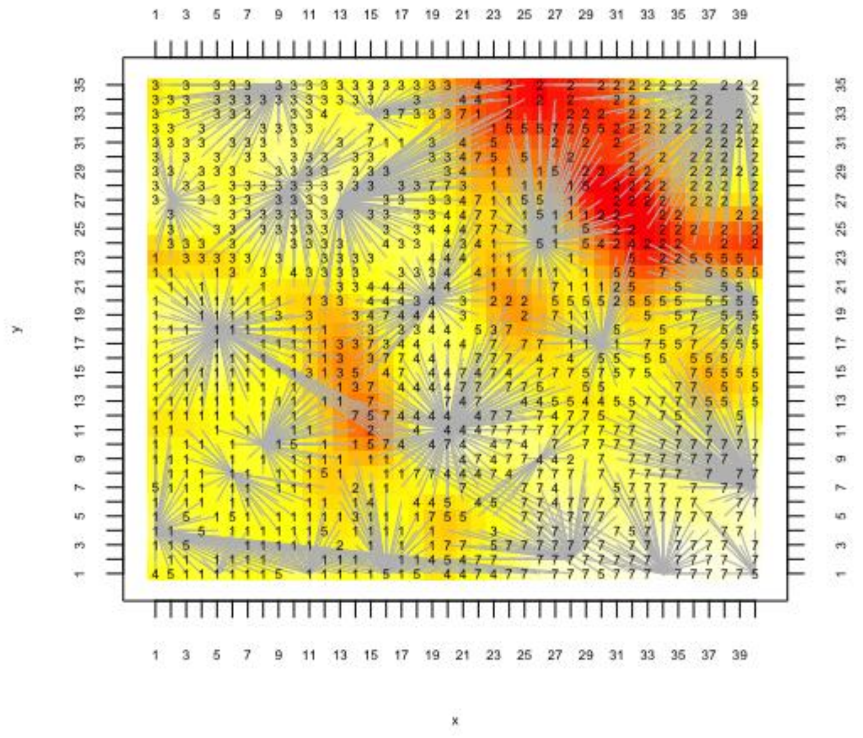


Figure 21. Starburst representation of ConvAE_SOM in ‘Landsat Satellite’

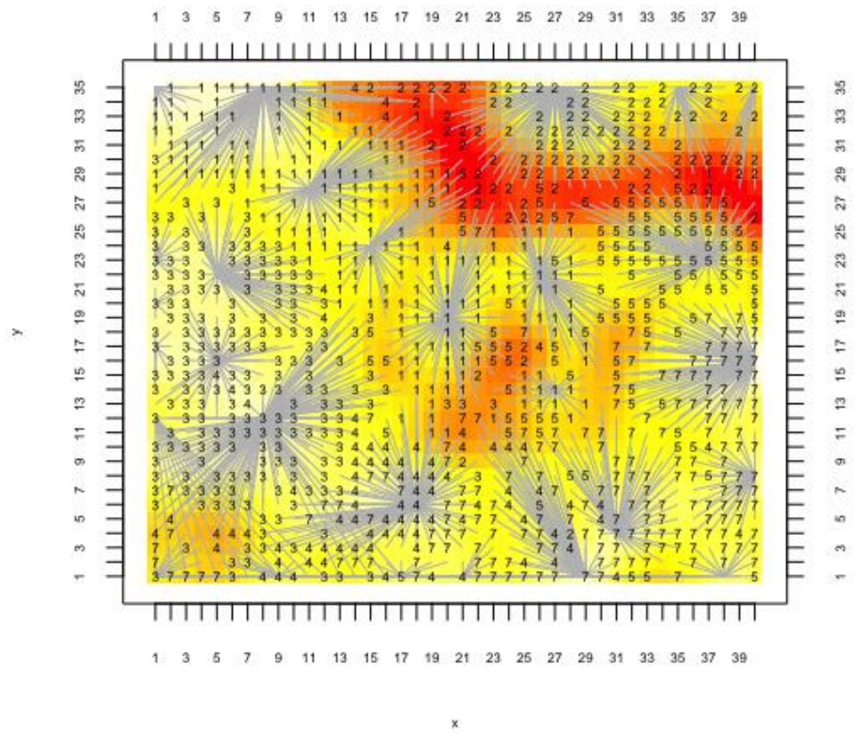


Figure 22. Starburst representation of SOM with unencoded data in ‘Landsat Satellite’

I implemented a 40×35 ConvAE_SOM compared with the SOM with unencoded data. I trained the models with 400,000 iterations. From the starburst representations shown in Figure 21 and Figure 22, the number of the identified clusters is almost the same and the visible starburst lines span in a similar way, which shows that the clustering structure is nearly the same. Therefore, the encoded data has a similar structure to the original data, and both structures are successfully discovered by the SOM. It also suggests that we can trust the encoded data as the input of the SOM.

4.3 'MNIST' Experiment Results

4.3.1 Loss of AEs

I plotted the loss and visible reconstruction results of each model, which are shown in Figure 23 – Figure 27.

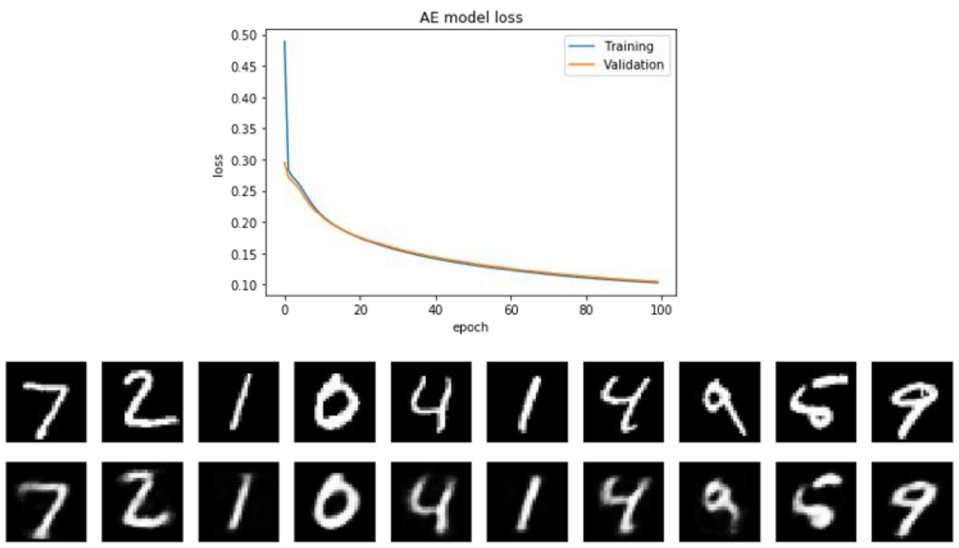


Figure 23. Loss of AE and reconstruction result

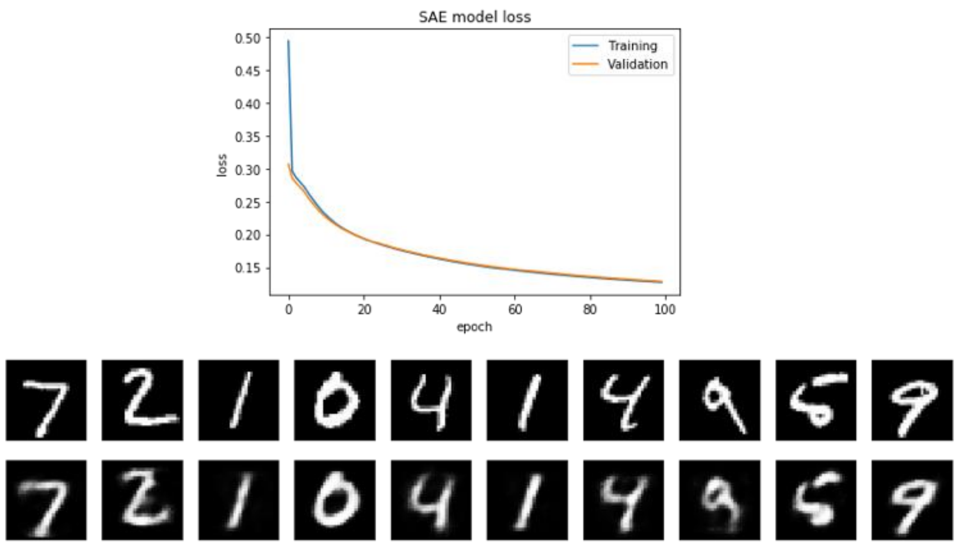


Figure 24. Loss of SAE and reconstruction result

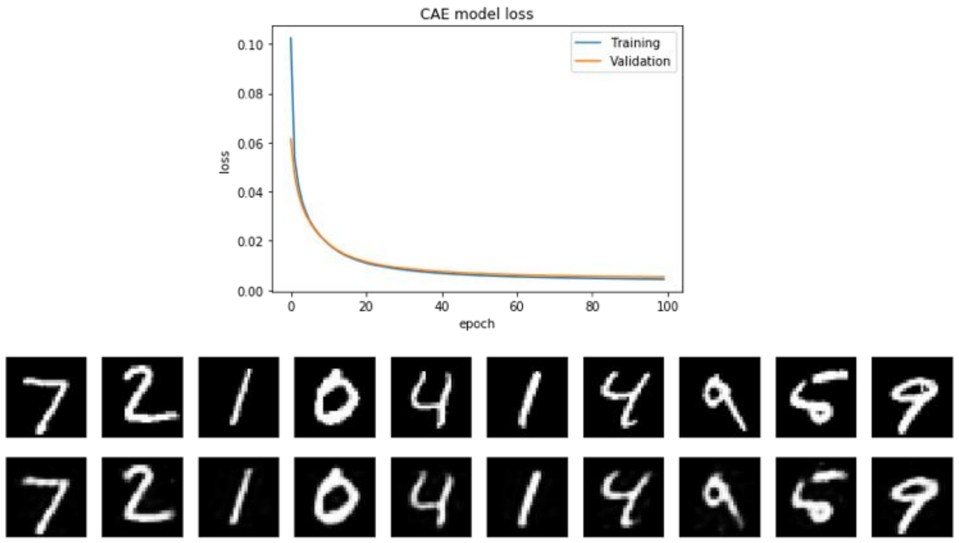


Figure 25. Loss of CAE and reconstruction result

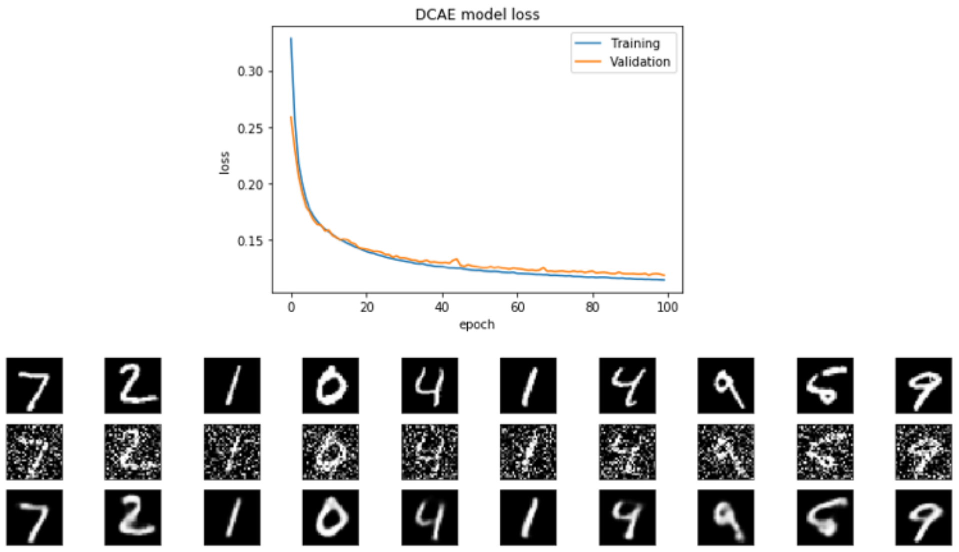
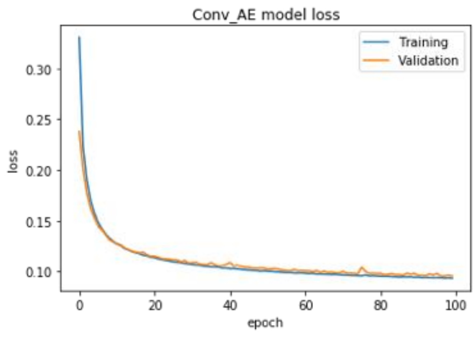


Figure 26. Loss of DCAE and reconstruction result



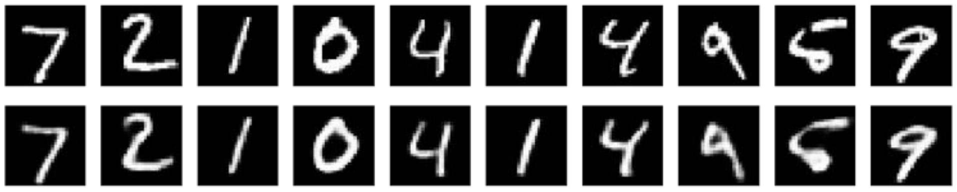


Figure 27. Loss of ConvAE and reconstruction result

The training loss and validation loss of each model show that all the models were trained well after 100 epochs. All the AEs reconstruct the original input. Judging from the visible results, DCAE and ConvAE did a better job.

4.3.2 SOM Models Results

Similarly, I plot the convergence accuracy, embedding accuracy, and estimated topographic accuracy of each model (Figure 28, Figure 29). I scaled the x-axis as log base 2.

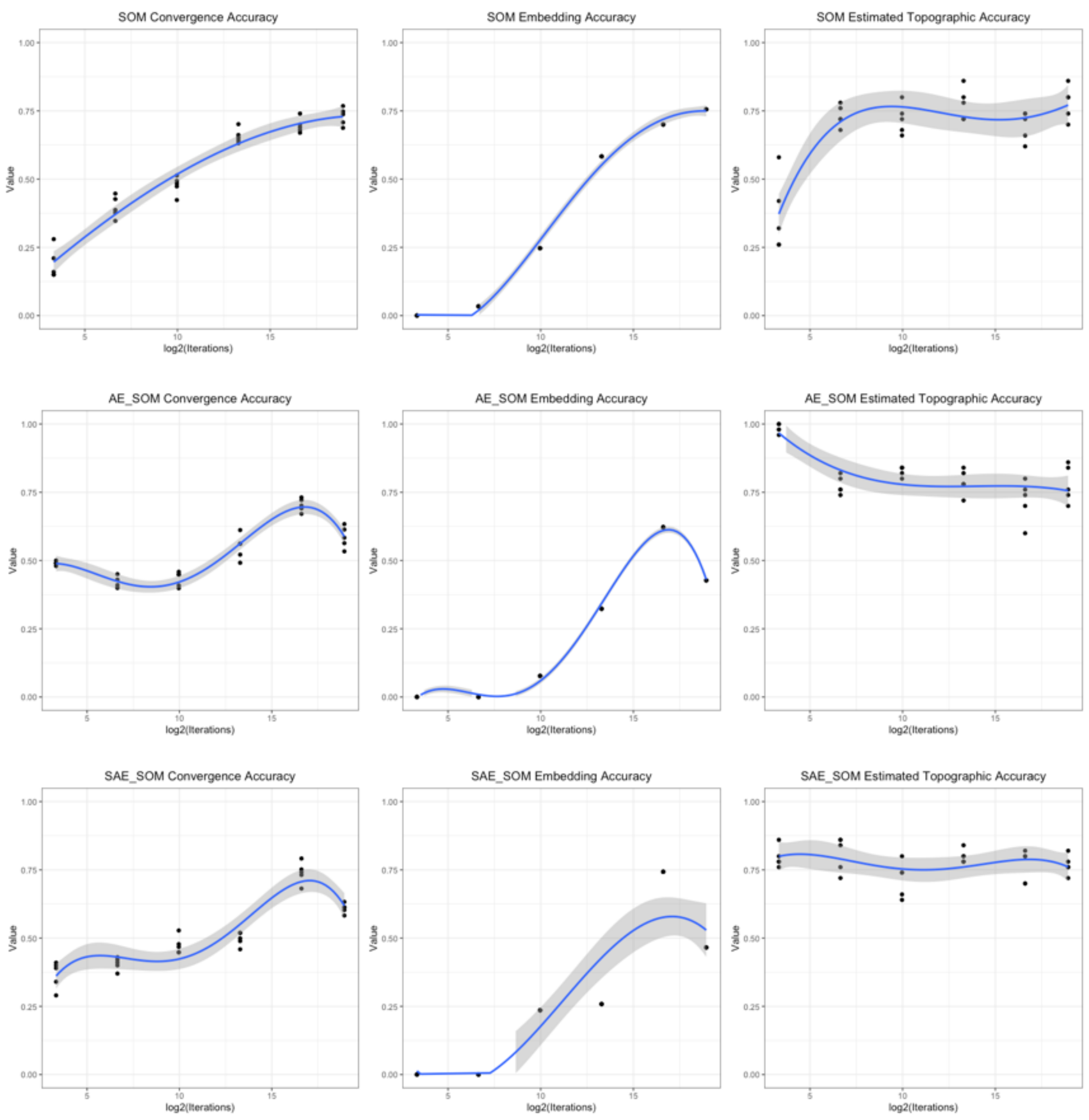


Figure 28. SOM, AE_SOM, SAE_SOM model quality measures in 'MNIST'

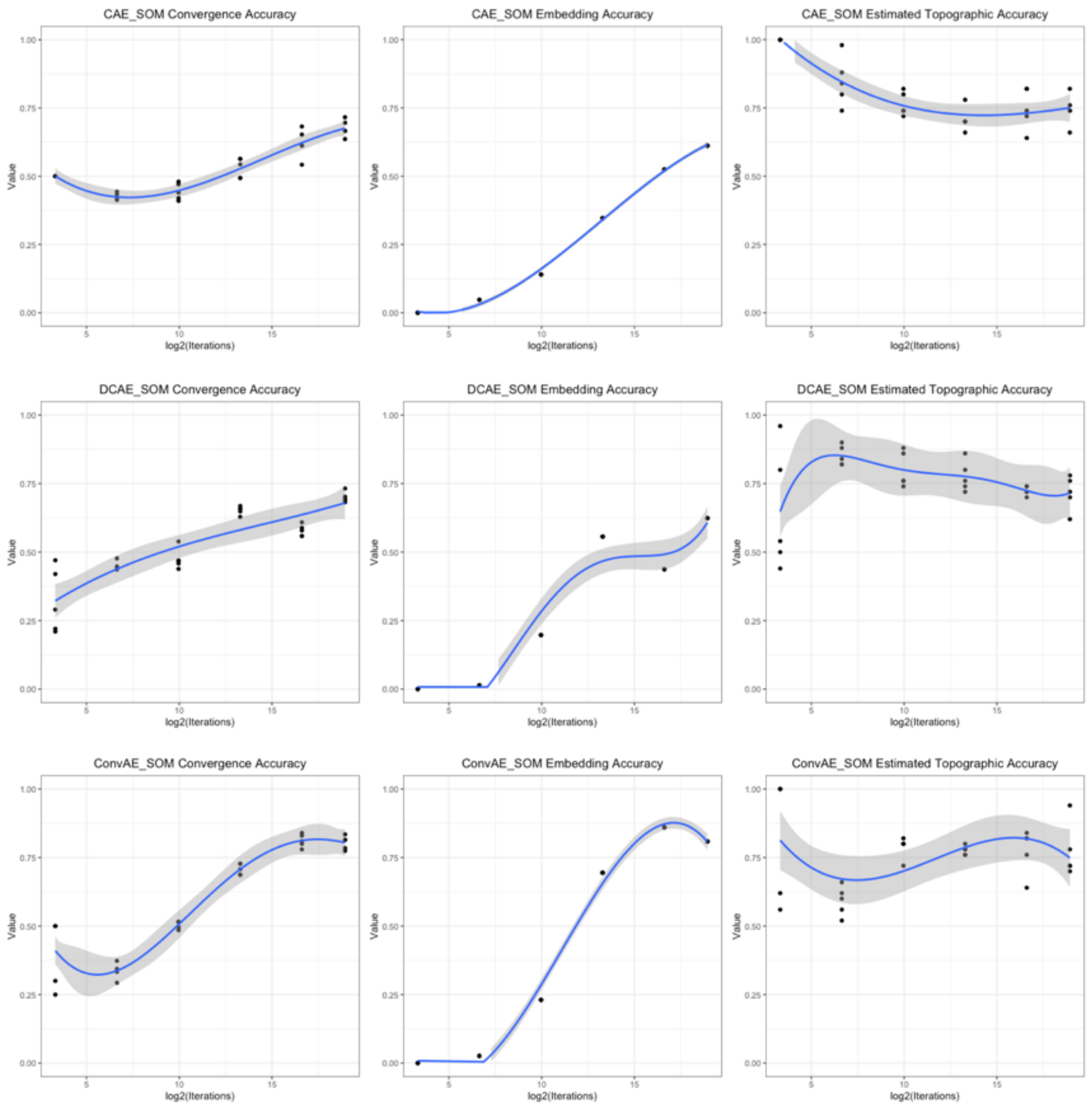


Figure 29. CAE_SOM, DCAE_SOM, ConvAE_SOM model quality measures in ‘MNIST’

To achieve better embedding accuracy, I chose a larger map size, which contains 1600 neurons. As a result, the topographic accuracy in all six models all exhibits a flat trend starting from the small number of iterations. The best clustering results are achieved by using ConvAE. It is not surprising that such AE performs best since convolutional operators capture the local features in images, which are the most important and informative ones for identification.

Generally speaking, to cluster MINST dataset by SOM is challenging as the data have high dimensionality and consist of plenty of zeros or near-zeros. This causes most of the features, namely the pixel values, which are not significant. While most AEs (except ConvAE) do not show significant improvement in clustering, they do provide a low dimensional representation containing intrinsic features and help to reduce training time in SOM.

4.3.3 Clustering Result Representation

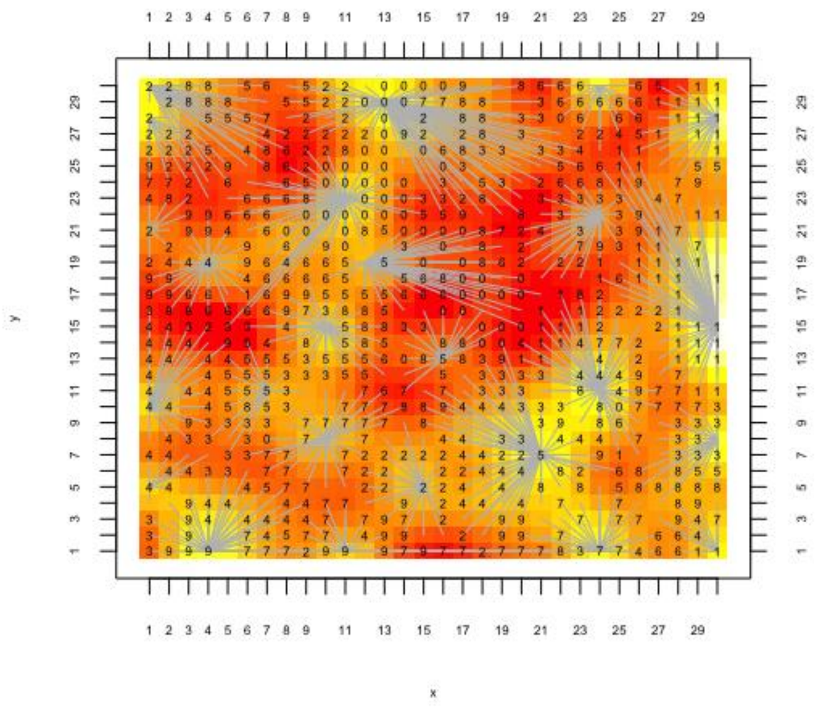


Figure 30. Starburst representation of ConvAE_SOM in ‘MNIST’

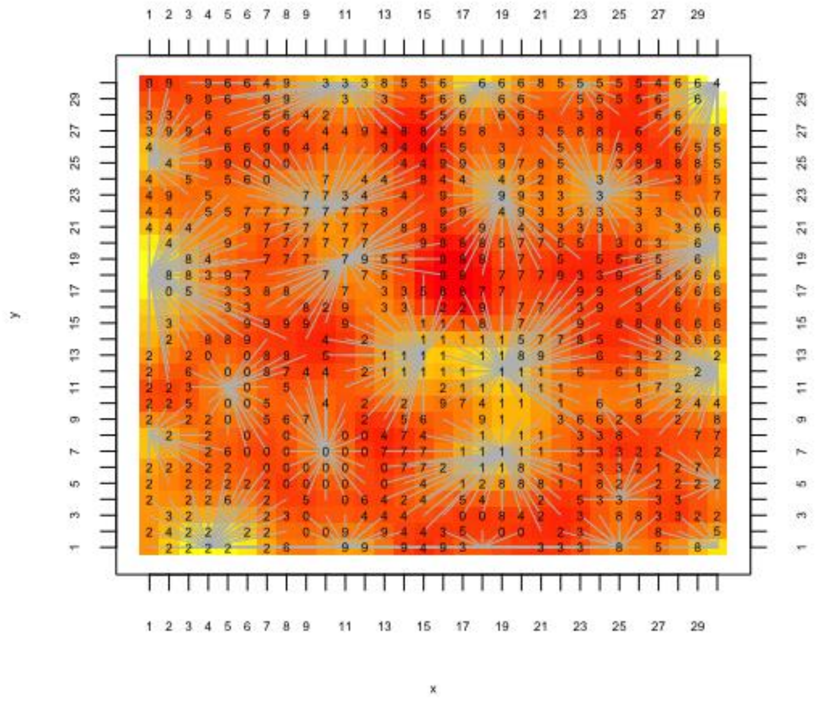


Figure 31. Starburst representation of SOM with unencoded data in ‘MNIST’

I implemented a 30×30 ConvAE_SOM trained with 100,000 iterations. In Figure 30, although the clustering results are not as good as the other two data sets, the model still achieved a reasonable cluster of some easily distinguishable digits. Compared with the starburst representation shown in Figure 31, the ConvAE_SOM shows a close clustering structure as the SOM with unencoded data because the number of the identified clusters are almost the same and the visible starburst lines span similarly. It indicates that the encoded data has a similar structure to the original data, and both structures are successfully discovered by the SOM. It also suggests that we can trust the encoded data as the input of the SOM.

CHAPTER 5

Conclusion

The objective of this research is to find answers for the following two questions, 1) for one data set, what kind of AE performs best in improving the performance of the underlying SOM, 2) whether the selection of AEs in conjunction with SOMs is data-dependent or not. According to the experiment results, we can see that nearly all AEs at least improve the performance of SOM. They also bring original data to a lower dimension representation, which let the training process become efficient. The CAE performed excellently in the synthetic data set. The ConvAE shows an outstanding performance in image-related data set. The DAE works well with the real-word data with noise. The SAE did not show good results in the three chosen data sets, which may be due to that data do not have the sparse property. Hence, the selection of the AEs depends on the property of data, based on the features of a data set to select an appropriate AE could help the SOM obtain a better clustering result.

Interestingly, many embedding accuracy figures have a peak value after a certain number of iterations. This could arise from that the neurons start to learn a finer-scale cluster; therefore, the embedding accuracy drops down a little. I suspect it will rise again until adequately learning an even finer scale in the future. To the end, each neuron is a cluster itself and the embedding accuracy approaches 1.

5.1 Future Work

Firstly, it is worth studying when the peak value of embedding accuracy comes out, which may help train a model with appropriate training iterations. For now, we could see that the embedding accuracy oscillates after the peak value, but I do not know the definite trend in the future. Training the model with much more iterations in the featured study will help discover the embedding accuracy variate trend and find the relationships between the peak value and training iterations.

Secondly, it is suggested to compare the SOM performance by using different dimensionality encoded data as input. In this research, I only encoded the original data into one type of dimensionality. Test different encoding dimensionality to see whether the encoding degree will affect the SOM clustering result could yield more interesting insight.

Additionally, there are still some other variations of AEs such as variational autoencoder and stacked autoencoder, which could be emphasized in the future study.

LIST OF REFERENCE

[1] G. E. Hinton and R. R. Salakhutdinov, “Reducing the Dimensionality of Data with Neural Networks,” *Science*, vol. 313, no. 5786, pp. 504–507, Jul. 2006.

[2] T. Kohonen, *Self-Organizing Maps*, 3rd ed. Berlin Heidelberg: Springer-Verlag, 2001.

[3] D. Rajashekar, “One-class learning with an Autoencoder Based Self Organizing Map,” Mar. 2017.

[4] M. Pesteie, P. Abolmaesumi, and R. Rohling, “Deep Neural Maps,” *ArXiv181007291 Cs Stat*, Oct. 2018.

[5] K. S. Pollard and M. J. van der Laan, “Cluster Analysis of Genomic Data,” in *Bioinformatics and Computational Biology Solutions Using R and Bioconductor*, R. Gentleman, V. J. Carey, W. Huber, R. A. Irizarry, and S. Dudoit, Eds. New York, NY: Springer New York, 2005, pp. 209–228.

[6] K. W. Govek, V. S. Yamajala, and P. G. Camara, “Clustering-independent analysis of genomic data using spectral simplicial theory,” *PLOS Comput. Biol.*, vol. 15, no. 11, p. e1007509, Nov. 2019.

[7] W. Jang and M. Hendry, “Cluster analysis of massive datasets in astronomy,” *Stat. Comput.*, vol. 17, no. 3, pp. 253–262, Sep. 2007.

[8] Y. Zhang and Y. Zhao, “Automated Clustering Algorithms for Classification of Astronomical Objects,” *Astron. Astrophys.*, vol. 422, no. 3, pp. 1113–1121, Aug. 2004.

[9] L. Hamel, “SOM Quality Measures: An Efficient Statistical Approach,” in *Advances in Self-Organizing Maps and Learning Vector Quantization*, Cham, 2016, pp. 49–59.

[10] Y. Cheng, “Convergence and Ordering of Kohonen’s Batch Map,” *Neural Comput.*, vol. 9, no. 8, pp. 1667–1676, Nov. 1997.

[11] L. Hamel, “VSOM: Efficient, Stochastic Self-organizing Map Training,” in *Intelligent Systems and Applications*, vol. 869, K. Arai, S. Kapoor, and R. Bhatia, Eds. Cham: Springer International Publishing, 2019, pp. 805–821.

[12] L. Hamel, B. Ott, G. Breard, R. Tatoian, and V. Gopu, *popsom: Functions for Constructing and Evaluating Self-Organizing Maps*. 2019.

[13] J. Schmidhuber, “Deep Learning in Neural Networks: An Overview,” *Neural Netw.*, vol. 61, pp. 85–117, Jan. 2015.

[14] D. H. Ballard, “Modular Learning in Neural Networks,” in *AAAI*, 1987.

[15] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. The MIT Press, 2016.

[16] B. A. Olshausen and D. J. Field, “Sparse coding with an overcomplete basis set: A strategy employed by V1?,” *Vision Res.*, vol. 37, no. 23, pp. 3311–3325, Dec. 1997.

[17] H. Lee, *Unsupervised feature learning via sparse hierarchical representations*. 2010.

[18] P. Domingos, *The master algorithm: How the quest for the ultimate learning machine will remake our world*. New York, NY, US: Basic Books, 2015.

[19] A. Makhzani and B. Frey, “k-Sparse Autoencoders,” *ArXiv13125663 Cs*, Dec. 2013.

[20] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, “Extracting and composing robust features with denoising autoencoders,” in *ICML*, 2008.

[21] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, “Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion,” *J Mach Learn Res*, vol. 11, pp. 3371–3408, Dec. 2010.

[22] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio, “Contractive Auto-Encoders: Explicit Invariance During Feature Extraction,” in *ICML*, 2011.

[23] G. Alain and Y. Bengio, “What Regularized Auto-encoders Learn from the Data-generating Distribution,” *J Mach Learn Res*, vol. 15, no. 1, pp. 3563–3593, Jan. 2014.

[24] X. Guo, X. Liu, E. Zhu, and J. Yin, “Deep Clustering with Convolutional Autoencoders,” in *Neural Information Processing*, 2017, pp. 373–382.

[25] F. Li, H. Qiao, B. Zhang, and X. Xi, “Discriminatively Boosted Image Clustering with Fully Convolutional Auto-Encoders,” *ArXiv170307980 Cs*, Mar. 2017.

[26] F. Chollet, *Keras, GitHub. https://github.com/fchollet/keras*. 2015.

[27] M. Abadi *et al.*, “TensorFlow: A System for Large-scale Machine Learning,” in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, Berkeley, CA, USA, 2016, pp. 265–283.

[28] P. Fränti and S. Sieranoja, “K-means properties on six clustering benchmark datasets,” *Appl. Intell.*, vol. 48, no. 12, pp. 4743–4759, Dec. 2018.

[29] P. Franti, O. Virmajoki, and V. Hautamaki, “Fast Agglomerative Clustering Using a k-Nearest Neighbor Graph,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 11, pp. 1875–1881, Nov. 2006.

[30] D. Dua and E. Karra Taniskidou, “‘UCI Machine Learning Repository,’ Irvine, CA: University of California, School of Information and Computer Science,” 2019. [Online]. Available: https://archive.ics.uci.edu/ml/citation_policy.html.

[31] Y. LeCun and C. Cortes, “MNIST handwritten digit database,” 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/.

[32] K. Kiviluoto, “Topology preservation in self-organizing maps,” in *Proceedings of International Conference on Neural Networks (ICNN’96)*, 1996, vol. 1, pp. 294–299 vol.1.

[33] L. Yuan, “Implementation of Self-Organizing Maps with Python,” *Open Access Masters Theses,* Jan. 2018.

[34] A. kristiadi, “Deriving Contractive Autoencoder and Implementing it in Keras - Agustinus Kristiadi’s Blog.” [Online]. Available: http://wiseodd.github.io/techblog/2016/12/05/contractive-autoencoder/.

BIBLIOGRAPHY

"About Keras Models - Keras Documentation." Accessed June 10, 2019. https://keras.io/models/about-keras-models/.

Ackermann, Nils. "Introduction to 1D Convolutional Neural Networks in Keras for Time Sequences." Accessed June 18, 2019. https://blog.goodaudience.com/introduction-to-1d-convolutional-neural-networks-in-keras-for-time-sequences-3a7ff801a2cf.

Alain, Guillaume, and Yoshua Bengio. "What Regularized Auto-Encoders Learn from the Data-Generating Distribution." *J. Mach. Learn. Res.* (January 2014): 3563–3593.

Baldi, Pierre. "Autoencoders, Unsupervised Learning, and Deep Architectures," 2012, 14.

Ballard, Dana H. "Modular Learning in Neural Networks." In *AAAI*, 1987.

Breard, Gregory. "Evaluating Self-Organizing Map Quality Measures as Convergence Criteria." *Open Access Master's Theses*, January 1, 2017.

Brown, David H. "Cartogram Data Projection for Self-Organizing Maps." *Dissertations and Master's Theses (Campus Access)*, January 1, 2012, 1–101.

"Building Autoencoders in Keras." Accessed June 10, 2019. https://blog.keras.io/building-autoencoders-in-keras.html.

Cheng, Yizong. "Convergence and Ordering of Kohonen's Batch Map." *Neural Computation* 9, no. 8 (November 1, 1997): 1667–76.

Cheng, Zhengxue, Heming Sun, Masaru Takeuchi, and Jiro Katto. "Deep Convolutional AutoEncoder-Based Lossy Image Compression." *ArXiv:1804.09535 [Cs]*, April 25, 2018.

DataCamp Community. "Keras Autoencoders: Beginner Tutorial," Accessed June 9, 2019. https://www.datacamp.com/community/tutorials/autoencoder-keras-tutorial.

Domingos, Pedro. *The Master Algorithm: How the Quest for the Ultimate Learning Machine Will Remake Our World*. Basic Books, 2015.

Franti, P., O. Virmajoki, and V. Hautamaki. “Fast Agglomerative Clustering Using a K-Nearest Neighbor Graph.” *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28, no. 11 (November 2006): 1875–81.

Fränti, Pasi, and Sami Sieranoja. “K-Means Properties on Six Clustering Benchmark Datasets.” *Applied Intelligence* 48, no. 12 (December 1, 2018): 4743–59.

GitHub. “Wiseodd/Hipsternet.” Accessed September 5, 2019. https://github.com/wiseodd/hipsternet.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2016.

Guo, Xifeng, Xinwang Liu, En Zhu, and Jianping Yin. “Deep Clustering with Convolutional Autoencoders.” In *ICONIP*, 2017.

Hamel, Lutz, Benjamin Ott, Gregory Breard, Robert Tatoian, and Vishakh Gopu. *Popsom: Functions for Constructing and Evaluating Self-Organizing Maps* (version 4.2.1), 2019.

Hamel, Lutz. “SOM Quality Measures: An Efficient Statistical Approach.” In *Advances in Self-Organizing Maps and Learning Vector Quantization*, 2016, 49–59.

Hamel, Lutz. “VSOM: Efficient, Stochastic Self-Organizing Map Training.” In *Intelligent Systems and Applications*, 2019.

Hinton, G. E., and R. R. Salakhutdinov. “Reducing the Dimensionality of Data with Neural Networks.” *Science* 313, no. 5786 (July 28, 2006): 504–7.

Khandelwal, Renu. “Deep Autoencoder Using Keras.”. Accessed June 9, 2019. https://medium.com/datadriveninvestor/deep-autoencoder-using-keras-b77cd3e8be95.

Kiviluoto, K. “Topology Preservation in Self-Organizing Maps.” In *Proceedings of International Conference on Neural Networks (ICNN’96)*, 1:294–99 vol.1, 1996.

Kohonen, Teuvo. *Self-Organizing Maps*. 3rd ed. Springer Series in Information Sciences. Berlin Heidelberg: Springer-Verlag, 2001.

LeCun, Y., and C. Cortes. “MNIST Handwritten Digit Database,” 2010. Accessed June 15. http://yann.lecun.com/exdb/mnist/.

Lee, Honglak. *Unsupervised Feature Learning via Sparse Hierarchical Representations*, 2010.

Li, Fengfu, Hong Qiao, Bo Zhang, and Xuanyang Xi. “Discriminatively Boosted Image Clustering with Fully Convolutional Auto-Encoders.” *ArXiv:1703.07980 [Cs]*, March 23, 2017.

Makhzani, Alireza, and Brendan Frey. “K-Sparse Autoencoders.” *ArXiv:1312.5663 [Cs]*, December 19, 2013.

Olshausen, Bruno A., and David J. Field. “Sparse Coding with an Overcomplete Basis Set: A Strategy Employed by V1?” *Vision Research* 37 (December 1, 1997): 3311–25.

Ott, Benjamin H. “A Convergence Criterion for Self-Organizing Maps.” *Dissertations and Master’s Theses (Campus Access)*, January 1, 2012, 1–70.

Pesteie, Mehran, Purang Abolmaesumi, and Robert Rohling. “Deep Neural Maps.” 2018.

Rajashekar, Deepthi. “One-Class Learning with an Autoencoder Based Self Organizing Map,” March 27, 2017.

Rifai, Salah, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. "Contractive Auto-Encoders: Explicit Invariance During Feature Extraction." In *ICML*, 2011.

Schmidhuber, Juergen. "Deep Learning in Neural Networks: An Overview." *Neural Networks* 61 (January 2015): 85–117.

Tatoian, Robert, and Lutz Hamel. "Self-Organizing Map Convergence:" *International Journal of Service Science, Management, Engineering, and Technology* 9, no. 2 (April 2018): 61–84.

"UCI Machine Learning Repository: Statlog (Landsat Satellite) Data Set." Accessed September 21, 2019. https://archive.ics.uci.edu/ml/datasets/Statlog+%28Landsat+Satellite%29.

Vincent, Pascal, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. "Extracting and Composing Robust Features with Denoising Autoencoders." 1096–1103. Helsinki, Finland: ACM Press, 2008.

Vincent, Pascal, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. "Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion." *J. Mach. Learn. Res.* 11 (December 2010): 3371–3408.

Yuan, Li. "Implementation of Self-Organizing Maps with Python." *Open Access Master's Theses*, January 1, 2018.