UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Heitor Rapela Medeiros

Deep Clustering Self-Organizing Maps with Relevance Learning

Recife
2020

Heitor Rapela Medeiros

Deep Clustering Self-Organizing Maps with Relevance Learning

A M.Sc. Dissertation presented to the Center of Informatics of Federal University of Pernambuco in partial fulfillment of the requirements for the degree of Master of Science in Computer Science.

Concentration Area: Computational Intelligence
Advisor: Hansenclever F. Bassani

Recife
2020

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

M488d Medeiros, Heitor Rapela
Deep clustering self-organizing maps with relevance learning / Heitor Rapela Medeiros. – 2020.
82 f.: il., fig., tab.

Orientador: Hansenclever de França Bassani.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2020.
Inclui referências.

1. Inteligência computacional. 2. Aprendizagem. I. Bassani, Hansenclever de França (orientador). II. Título.

006.31 CDD (23. ed.) UFPE - CCEN 2020 - 208

Heitor Rapela Medeiros

“Deep Clustering Self-Organizing Maps with Relevance Learning”

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Aprovado em: 16/09/2020.

BANCA EXAMINADORA

_____

Prof. Dr. Tsang Ing Ren
Centro de Informática / UFPE

_____

Prof. Dr. Guilherme de Alencar Barreto
Departamento de Engenharia de Teleinformática / UFC

_____

Prof. Dr. Hansenclever de França Bassani
Centro de Informática / UFPE
(Orientador)

I dedicate this thesis to my family and my wife for all support. I would also like to dedicate this work to my coworkers, professors, and friends that helped me a lot.

ACKNOWLEDGEMENTS

My journey as a scientist/engineer started when I was younger, and I followed my grandfather disassembling the videotape, I liked a lot the mechanical and electronic part. After some time, my grandfather passed away, but I always remember his teachings on how to fix things. So, all effort that I made to this work was because I would like to be awesome like he was. I remembered that one day I got his wood, and I built a structure that I took care of like a robot inspired by the Medabots, a robot's cartoon.

When I was in high school, I was in doubt about mechatronics, computer science, computer engineering, and aeronautic engineering, but I chose computer engineering. I thought that this course would give me the opportunity to learn how to build a robot. During these days, my girlfriend, Lorenna, was trying to be approved in a public university in Pernambuco, and I liked to give her support to not give up on her dreams to be a doctor, and she also gave me a lot of support in my projects, inclusively in RoboCIn. After being approved for the computer engineer course at UFPE, I made several friends, and these friends helped me a lot to be stronger and better. My friends are amazing, and I participated in many projects with them like ArduRec, RoboCIn, Intel SBESC Competitions. It was really cool, and those projects gave me many opportunities to show my potential as an engineer and a scientist. For those who want to follow a hard work pathway, with many friends and projects, I like to think that "Genius is 1% inspiration, and 99% perspiration" by Thomas Edson, this phrase guide me a lot to continue my work. To the RoboCIn for the amazing projects, ideas, discussions, and meetings that we, as family, had together.

I would like to tell about this thesis experience, sometimes I think like Richard Feynman, because "What I cannot create, I do not understand", and during this research, I did many things that I did not understand well, but my coworkers helped me a lot with insightful discussions. Also, I would like to paraphrase Isaac Newton: "If I have seen further than others, it is by standing upon the shoulders of giants", because without the guidance of my advisor Hansenclever F. Bassani, and my coworker Pedro Braga, and seen further with their help, I could not see the pathway and use many ideas in collaboration with them to build this work. Here, I would like to thank you for the support and help in the ideas, coding, and nights without sleep.

The author of this thesis caveat all the success of the works to the insightful discussions and the hard work delivered by its collaborators. So, I would like to thank all coworkers for the fantastic effort to make possible this Dissertation and the articles get published. In special Hansenclever F. Bassani, Pedro H. Braga, Felipe Duque, and Aluizio F. Ribeiro. I would also like to thank CAPES for financial support of this work, and NVIDIA for the GPU's grant that helped in the final steps of this work.

To my family, I would like to thanks my mom, my father, and my brothers to be really amazing and take care of my sanity in this crazy world that we live in. I would also like to thank

you for my second and many other families that I have, e.g., Penha and Aunt Marcilene, and also, my grandmothers and grandfathers. And this one is for my wife: "Te amo visse!" and always keep a little beach day in your heart.

Thank you for being part of this journey, help me to do good scientific work in our country. So, let's keep working hard toward our goals!

ABSTRACT

In recent years, with the advancement of the internet, there has been an increase in data available. Alongside various data sources like sensors began to generate data of all kinds. Extracting useful information from data is a challenging problem in machine learning. The emerging focus on machine learning research has been the field of deep learning, which aims to learn multiple layers of abstraction that can be used to interpret data and perform complex tasks. In fact, the successful results of deep learning rely on the supervised field, which needs a large amount of labeled data. Unsupervised deep learning models have been proposed to deal with data without the requirement of annotations, which incorporate the data itself as a clue to guide the learning process. In that way, this thesis presents Deep Clustering Self-Organizing Map with Relevance Learning (DCSOM-RL), an unsupervised learning model capable of working with complex data, such as images and sound, while learning representations more suitable for clustering in latent spaces. The proposed approach combines deep learning architectures such as Autoencoders with a SOM layer with time-varying topology. The results show that the prototypes identified by DCSOM-RL represent frequent variations observed in the input data. For instance, the different ways to represent the input data. It can also bring insights about similarities between different categories or feature representations and which dimensions of the latent space capture important information. The neighborhood learned by the DCSOM-RL leads to smoother regions of transition between categories in the latent space. Although it does not present state of the art results in terms of evaluation metrics, the qualitative analysis shows that the model presents unique properties not available in other methods of Deep Clustering methods.

Keywords: Self-Organizing Maps. Unsupervised Learning. Deep Learning. Autoencoder.

RESUMO

Nos últimos anos, com o avanço da internet, houve um aumento na quantidade de dados disponíveis. Paralelamente, várias fontes de dados começaram a gerar dados de todos os tipos. Extrair informações úteis de dados é um problema desafiador na aprendizagem de máquina. O foco emergente na pesquisa da aprendizagem de máquina tem sido o campo do aprendizado profundo, que visa aprender várias camadas de abstrações que podem ser usadas para interpretar dados e realizar tarefas complexas. Na verdade, os resultados de sucesso da aprendizadem profunda dependem do campo supervisionado, que precisa de uma grande quantidade de dados rotulados. Modelos de aprendizagem profunda não supervisionados têm sido propostos para lidar com dados sem a necessidade de anotações, que incorporam os próprios dados como uma pista para orientar o processo de aprendizagem. Dessa forma, esta Dissertação apresenta o Deep Clustering Self-Organizing Map with Relevance Learning (DCSOM-RL), um modelo de aprendizagem não supervisionado capaz de trabalhar com dados complexos, como imagens e sons, enquanto aprende representações mais adequadas para agrupamento em espaços latentes. A abordagem proposta combina arquiteturas de aprendizagem profunda, como Autoencoders, com uma camada SOM com topologia variável no tempo. Os resultados mostram que os protótipos identificados pelo DCSOM-RL representam variações frequentes observadas nos dados de entrada. Por exemplo, as diferentes formas de representar um dado de entrada. Também pode trazer insights sobre semelhanças entre diferentes categorias ou representações de features e quais dimensões do espaço latente capturam informações importantes. A vizinhança aprendida pelo DCSOM-RL leva a regiões mais suaves de transição entre categorias no espaço latente. Embora não apresente resultados estado da arte em termos de métricas de avaliação, a análise qualitativa mostra que o modelo apresenta propriedades únicas não disponíveis em outros métodos de Deep Clustering.

Palavras-chave: Self-Organizing Maps. Aprendizagem Não Supervisionada. Aprendizagem Profunda. Autoencoder. Espaço Latente.

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

AE	*autoencoders*
AI	*artificial intelligence*
ALTSS-SOM	*adaptive local thresholds semi-supervised self-organizing map*
ANN	*artificial neural networks*
ARI	*adjusted rand index*
Batch-*semi-supervised self-organizing map* (SS-SOM)	*batch semi-supervised self-organizing map*
CE	*clustering error*
CIFAR10	*Canadian Institute For Advanced Research*
CLMSDC	*closed-loop multi-step deep clustering*
CNN	*convolutional neural networks*
CSOM	*convolutional self-organizing map*
DBN	*deep belief networks*
DC	*deep clustering*
DCSOM-RL	*deep clustering self-organizing map with relevance learning*
DE-SOM	*deep embedded self-organizing map*
DEC	*deep embedded clustering*
DL	*deep learning*
DNN	*deep neural networks*
DOC	*densitive-based optimal projective clustering*
DSOM	*deep self-organizing map*
DSSOM	*dimension selective self-organizing map*
ELBO	*evidence lower bound*
FCN	*fully-connected network*
GAN	*generative adversarial networks*
GPUs	*graphics processing units*
IIC	*invariant information clustering*
JDC	*joint deep clustering*
KL divergence	*Kullback–Leibler divergence*
LARFDSSOM	*local adaptive receptive field dimension selective self-organizing map*
LARFSOM	*local adaptive receptive field self-organizing map*
LHS	*latin hypercube sampling*

LVQ	*learning vector quantization*
ML	*machine learning*
MLP	*multilayer perceptron*
MSE	*mean square error*
MSSDC	*multi-step sequential deep clustering*
NMI	*normalized mutual information*
PCA	*principal component analysis*
PROCLUS	*PROjected CLUStering algorithm*
RBM	*restricted boltzmann machine*
SAE	*stacked autoencoder*
SGD	*stochastic gradient descent*
SIFT	*scale-invariant feature transform*
SOM	*self-organizing map*
SOM-VAE	*self organizing map variational autoencoder*
SOM-RL	*self-organizing map with relevance learning*
SSL	*semi-supervised learning*
SS-SOM	*semi-supervised self-organizing map*
SSSOM-RL	*semi-supervised self-organizing map with relevance learning*
SVHN	*Street View House Numbers*
UVOC	*unsupervised visual object categorization*
VAE	*variational autoencoders*
VQ	*vector quantization*
VQ-VAE	*vector quantised-variational autoencoder*

LIST OF SYMBOLS

Φ	Nodes indexes Space
Ψ	Prototypes space
α	Constant to weight the loss function
η	Learning rate of SOM
$\mathbb{R}$	Real numbers set
ε	A small number to avoid division by zero
ω	Relevance vector
$\mathbf{x}$	Input vector
$\mathscr{L}$	Loss symbol
∇	Gradient
$\text{sg}[\cdot]$	Gradient stopping operator used to set the gradient to 0.
k	Number of k-means clusters

CONTENTS

1 INTRODUCTION

In recent years, with the advancement of the internet, there has been an increase in the amount of data available. Alongside this, the so-called Internet of Things (IoT) emerged, in which various data sources such as cameras and other devices began to generate data that were used for progress in various areas of computing such as data acquisition, data compression, data storage, data transmission, and processing complex and high dimensional data (Vidal, 2011). The advancement of IoT brought the development of visualization techniques that allowed users to understand better how their data is organized and make decisions in more natural ways. Extracting meaningful information from raw data is a non-trivial activity, especially information from sensors from different perspectives and time resolutions, which is a challenging problem in *machine learning* (ML).

ML algorithms are commonly divided in three main areas: supervised learning, unsupervised learning, and reinforcement learning (Bishop, 2006). Supervised learning is a task-driven process, in which the training data comprises examples of the input vectors and their corresponding label vectors. In unsupervised learning, the task is driven by the data, in which the training data comprises only the input vector without the label vector. The Unsupervised Learning algorithm uses the data to create a better representation to solve tasks such as clustering or determine the distribution of data, known as density estimation, or to map the data from a high-dimensional space to a lower-dimensional space for tasks such as visualization. In Reinforcement Learning, algorithms learn to react to an environment, so it essentially calculates some reward for an action in the environment, and it has to decide if it was good or not to the agent. It discovers the solution through the process of trial and error (Bishop, 2006). More recently, the emerging focus of machine learning research has been the field of deep learning, which aims to learn multiple layers of abstraction that can be used to interpret data (Gubbi *et al.*, 2013). DL techniques have been using the intersection of these three main areas of ML to create better models in terms of generalization (Bengio *et al.*, 2007).

Due to the growth of structured data, benchmark datasets, *graphics processing units* (GPUs), techniques that have been proposed in the last decades have improved, and now larger models can be trained effectively and tested in real-world applications. Now, it is possible to train an *artificial neural networks* (ANN) with many layers. Therefore, more robust approaches are available for many important tasks that use ML, e.g., face detection to unlock smartphones or voice recognition in virtual assistant applications. DL is a powerful tool that creates a useful representation that can be used to perform complex tasks, such as, object detection (Ren *et al.*, 2015; Erhan *et al.*, 2014; Redmon & Farhadi, 2018), image classification (Krizhevsky *et al.*, 2012; Liang & Hu, 2015), audio processing (Lee *et al.*, 2009; McLoughlin *et al.*, 2015; Graves *et al.*, 2013), natural language processing (Kim, 2014; Kalchbrenner *et al.*, 2014), and many others tasks. The positive aspect of DL is that manual feature engineering has decreased with the advance of DL, minimizing errors introduced by handcrafted features. In return, powerful

machines are needed to compute complex machine learning operations with a significant amount of data and transform it into useful representations for the models. When applied to traditional ML competitions, such as ImageNet, DL approaches have overcome classical techniques and, in some cases, even human performance (Russakovsky *et al.*, 2015). Therefore, DL has become the standard approach to solving many hard tasks with the usage of data, such as the ones illustrated in Figure 1 and Figure 2, respectively, the detection of people in images and of robots for soccer competition.

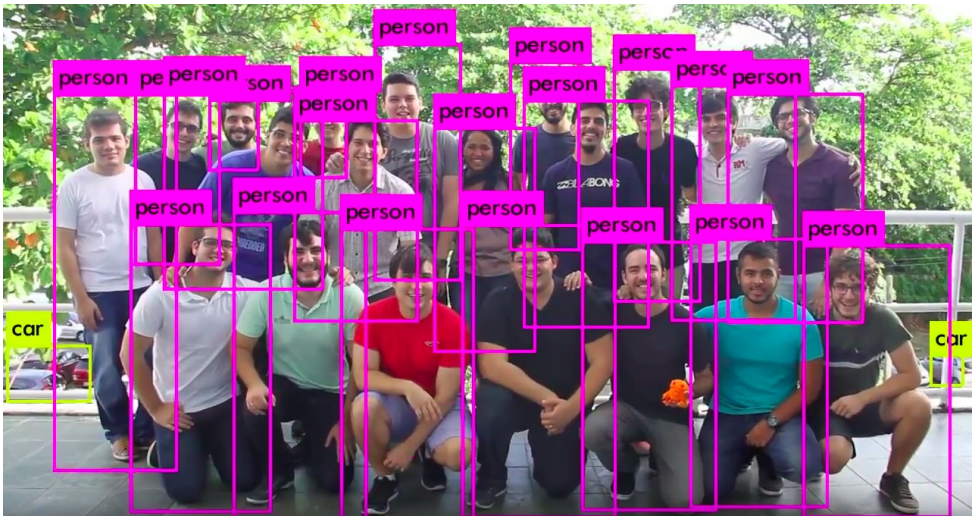


Figure 1: RoboCIn 2017 members detected using DL models (YOLO architecture pre-trained on Pascal Dataset). Image provided by the author.

Figure 2: Robot detection during a match of robot soccer competition. Pytorch YOLO architecture trained on images labeled by the author. Image provided by the author.

A large number of companies have been investing in DL techniques because it is practical to solve specific tasks and, as said before, generally outperform standard handcraft techniques.

The main success of DL techniques are in supervised learning tasks, but good labels are hard to obtain. Unsupervised learning models have been proposed to deal with data without the need for labels, methods that incorporate the data itself as a clue to guide the learning process. In Deep Learning, models like *autoencoders* (AE), *restricted boltzmann machine* (RBM), and *generative adversarial networks* (GAN) are being widely used, but they lack the relationship between the clusters and its neighborhood (topological preservation of the neighborhood), which is important to understand the relationship between different inputs and its features. A model that preserves the topology of the input data can manipulate the data without damage in its neighborhood. The brain can inspire these models because its information is organized in many places with some relations in such a way that maps with topological order (Haykin, 1994).

Inspired by the neurophysiology of the brain, the SOM was proposed by Kohonen (1990). SOM networks can map a high-dimensional distribution in a smaller regular grid, managing to compress information while preserving the topology of the input data. With this, SOM have been used for clustering tasks. Variations of SOM networks were proposed for subspace clustering problems with improvements in parameterization and time-varying topology. Unlike common clustering algorithms, these SOM-based models do not need to know the exact number of clusters beforehand. However, these models cannot adequately handle more complex data structures, such as images.

Many tasks that depend on data can become harder or easier, depending on how that data, e.g., an image, is represented. For example, representing a circle or curve in an image could make detecting a soccer ball easier than looking at each image pixel. A good representation is the one that is capable of making the subsequent task easier (Goodfellow *et al.*, 2016). In computer vision, algorithms like canny edge detector or color image segmentation can extract low-level features such as borders with filters, and then use transformations in the images to detect more common patterns. A traditional way is to use abstractions of pixel representations such as the presence of borders, detection of specific shapes, and identification of categories. In the end, all this information must be grouped in order to understand the scenario (Bengio *et al.*, 2009). DL techniques have been shown as a powerful tool to create representations useful for solving tasks. The work done by Lawrence *et al.* (1997) introduces a way to combine SOM with *convolutional neural networks* (CNN) to deal with images, since SOM do not work well directly with images like CNN, providing good evidence that these two models can be coupled to solve complex problems.

The main idea explored in this work is to investigate the use of more SOM-based methods, exploring its properties of clustering and topologically ordered mapping to guide the process of learning a good representation in the latent space obtained by deep neural networks. This will allow learning mappings from a complex input space to a simpler latent space in which SOM performs its functions.

To work toward our goal and bring SOM for state-of-art deep learning scenario, the SOM needs to work with high-dimensional data like images, which do not have good results when

applied directly to SOM and comparing with outstanding results given by CNN models. We also need to make a deep learning SOM layer and open-source it for the science community validate and deploy SOM models, a common practice in the deep learning field. Finally, we think that the SOM have prototype learning properties that could guide the feature learning, so we would like to test this point in the thesis.

Considering what has been set out, the main objectives of this thesis are listed below:

- Enable the usage of images with self-organizing maps by combining it with supervised DL approaches for learning representations.
- Develop a SOM layer compatible with moderns DL frameworks, like PyTorch, and evaluate it in the unsupervised learning scenario.
- Apply the SOM layer to guide the organization of the latent space of an autoencoder to achieve a fully unsupervised end-to-end learning model for representation learning and clustering.

The first objective is achieved with the proposal of a framework combining feature extraction from images with a pre-trained DL model combined with LARFDSSOM. Later on, the second objective is achieved with a fully unsupervised learning model, in which the pre-trained model is replaced by an autoencoder and a SOM layer is trained with the features provided by the encoder. Finally, the third objective is accomplished with the development of a system capable of backpropagating the unsupervised error of SOM layer to the encoding layers. The full framework composes AE with SOM-based models embedded in its latent space and is called deep clustering self-organizing map with relevance learning (DCSOM-RL). The DCSOM-RL is an unsupervised learning model capable of cluster and reconstruct complex data while learns more suitable representations for clustering in latent spaces. The results show that the prototypes identified by DCSOM-RL represent frequent variations observed in the input data. For instance, the different ways to represent the input data. It can also bring insights about similarities between different categories or feature representations and which dimensions of the latent space capture important information. The neighborhood learned by the DCSOM-RL leads to smoother regions of transition between categories in the latent space. Although it does not present the state of the art results in terms of evaluation metrics, the qualitative analysis shows that the model presents unique properties not available in other methods of Deep Clustering methods.

The rest of this document is organized as follows: Chapter 2 presents essential concepts related to the areas where this work is inserted in. Chapter 3 introduces work in the literature that is more closely related to the ideas developed to build the proposed models. Chapter 4 describes in detail the proposed models. Later on, in Chapter 5, the experimental setup, methodology, obtained results, and comparisons will be discussed in order to evaluate the models proposed. Finally, Chapter 6 concludes this thesis by summarizing the results obtained and indicating future directions and practical applications for the proposed models.

2 THEORETICAL BACKGROUND

In the current chapter, the theoretical background is presented to consolidate important concepts required for the understanding of the research. First, Section 2.1 introduces clustering, subspace clustering, and how relevance learning can be used to deal with subspace clustering problems. Second, in Section 2.2, an overview of SOM is provided. Third, Section 2.3 describes advances and incremental modifications on top of SOM to work with dynamic topology and relevance learning. Finally, in Section 2.4, the concepts of deep representations and *autoencoders* are explained.

2.1 CLUSTERING AND SUBSPACE CLUSTERING

The goal of clustering is to separate a set of data points into groups based on similarities between the data points. Therefore, clustering is considered to be the most natural way of summarizing and organizing data.

In clustering, a collection of prototype vectors, which may not necessarily be a member of the data set, is determined by minimizing a distance-based object function. Considering a training set $X = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$, where $\mathbf{x}_i = (x_i^{(1)}, ..., x_i^{(d)}) \in \mathbf{R}^d$ are the d-dimensional feature vectors. The *vector quantization* (VQ) codebook, a set of code vectors, which is denoted here as $C = \{\mathbf{c}_1, ..., \mathbf{c}_K\}$ where $K \ll N$, forms the clustering structure of those clustering models.

Subspace clustering is an extension of clustering that consists of identifying the clusters and relevant dimensions for each cluster. The subspace clustering tasks requires specialized methods to deal with the intersection of the clusters. The following definition for subspace clustering is proposed by Vidal (2011): Consider a collection of data points with a union of subspaces, as illustrated in Figure 3. Specifically, let $\left\{\mathbf{x_j} \in \mathbb{R}^{\mathbf{D}}\right\}_{j=1}^{N}$ be a given set of points drawn from an unknown union of $n \geq 1$ linear or affine subspaces $\{S_i\}_{i=1}^{n}$ of unknown dimensions $d_i = \dim(S_i), 0 < d_i < D, i = 1, \ldots, n$. The next Equation (2.1) define the subspaces:

$$S_i = \left\{x \in \mathbb{R}^D : x = \mu_i + U_i y\right\}, \quad i = 1, \ldots, n, \tag{2.1}$$

where $\boldsymbol{\mu}_i \in \mathbb{R}^D$ is an arbitrary point in subspace S_i that can be chosen as $\boldsymbol{\mu}_i \in \mathbb{R}^D$ for linear subspaces, $U_i \in \mathbb{R}^{D \times d_i}$ is a basis for subspace S_i , and $y \in \mathbb{R}^{d_i}$ is a low-dimensional representation for point $\boldsymbol{x}$. The goal of subspace clustering is to find the number of subspaces n, their dimensions $\{d_i\}_{i=1}^{n}$, the subspace bases $\{U_i\}_{i=1}^{n}$, the points $\{\boldsymbol{\mu}_i\}_{i=1}^{n}$, and the segmentation of the points according to the subspaces.

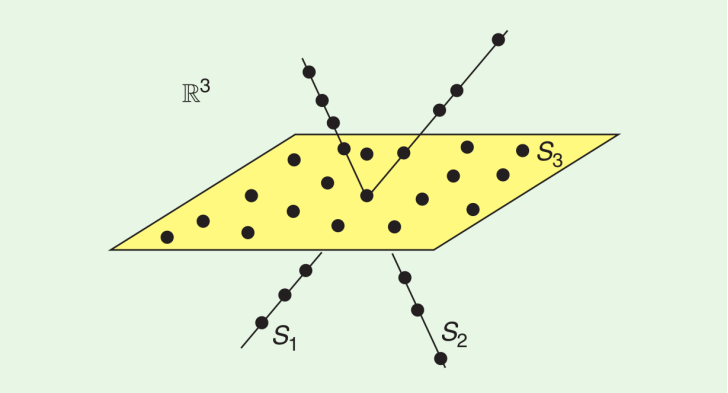


Figure 3: A set of sample points in $\mathbb{R}^3$ drawn from a union of three subspaces: two lines and a plane. Source: Vidal (2011).

In subspace clustering, the aim is to group the data simultaneously in multiple subspaces within a data-driven way. This data-driven process works to create clusters that separate better the intersection of useful features in the feature space. This problem appears in areas such as image segmentation, motion tracking, or gene expression determination (Vidal, 2011). Figure 4 illustrates an example of a toy problem that can be solved with subspace clustering, in which the red color feature is among one dimension, and the hexagon shape feature is among the other one, the intersection of this problem consist of the middle red hexagon and this property can not be fully disentangled by classical clustering algorithms that do not accomplish subspace clustering tasks. Hence, algorithms that perform subspace clustering should be capable of identifying and separating better the intersection of the clusters presented on the before-mentioned toy problem.

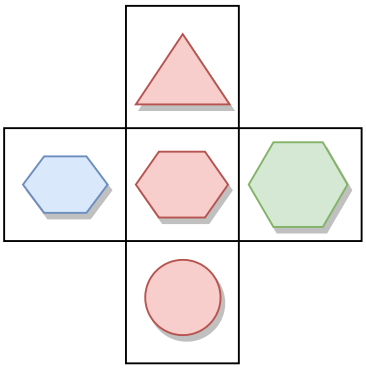


Figure 4: Subspace Clustering toy problem. The problem consists of two dimensions features: color and shape. Algorithms that perform subspace clustering works to identify and separate better the intersection of those dimensions.

2.2 SELF-ORGANIZING MAP (SOM)

A *self-organizing map* (SOM), proposed by Kohonen (1990), is a two-layered ANN with unsupervised learning. This network performs a mapping of a continuous input space to a discrete output space of a lower dimension, where the topological properties of the inputs are

preserved as best as possible. Figure 5 illustrates a SOM, with its two layers.

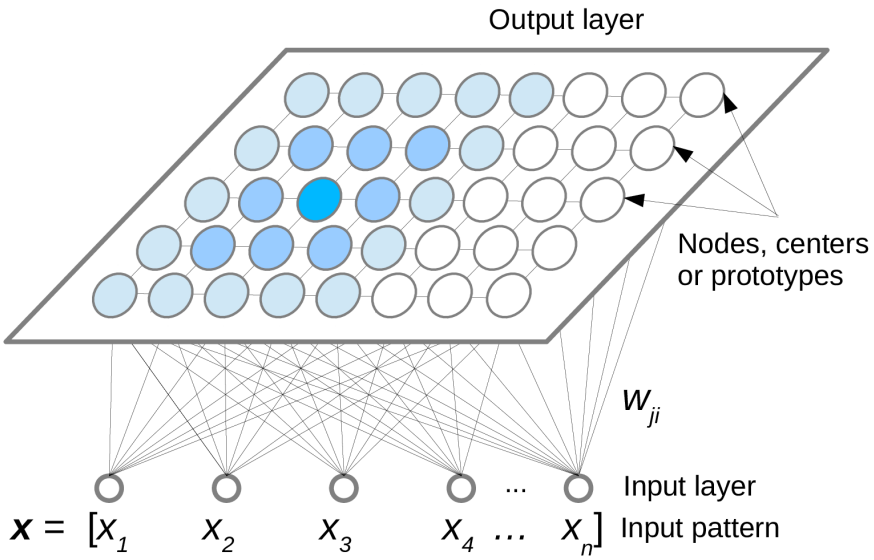


Figure 5: Illustration of a SOM, showing the input pattern (vector **x**, at the bottom), the nodes at the input layer, the nodes in the output layer with their lateral connections, and the weights w_{ji} that represents a connection between the i-th node in the input layer and the j-th node in the output layer. Source: Bassani & Araújo (2012).

In addition, as SOM have the ability to organize complex dimensional data into groups (clusters) according to their similarity relationships, in this case, the nodes are also called prototypes because they summarize a certain region of the input space. SOM does not require labeled samples, making it ideal for problems in which labels are unknown or undetermined (Haykin, 1994). This type of network is used for a variety of applications, such as data visualization (da Silva & Wunsch, 2017), speech recognition (Lokesh *et al.*, 2019), image analysis (Farooq *et al.*, 2017), industrial control processes (Cirera *et al.*, 2020).

In SOM, the neurons in the output layer (also called nodes or units) are organized in a regular structure, the most commonly used being the two-dimensional hexagonal or rectangular grid, as illustrated in Figure 6. Each neuron j is associated with a vector of weights, $\mathbf{c}_j$ [1], with the same dimension as the input data. The algorithm for training a SOM is a learning algorithm, in which the patterns are mapped to the neuron which weight vector is closest according to a distance metric, such as the Euclidean distance or Manhattan distance.

The operation of a SOM can be divided into two stages: training and mapping. Weight vectors can be initialized randomly or following the sampled patterns of the input dataset. Then, the following is repeated until convergence: randomly choose a pattern **x** from the input dataset and present it to the input layer. For each node in the output layer: the distance between the weights and the input pattern **x** is calculated using Equation (2.2), and with this, the weight vector of the winning neuron $i(\mathbf{x})$, which has the minimum distance, is updated.

[1] In LARFDSSOM, the notation **c** is used instead of **w** following the tradition of time-varying topology models.

$$i(\mathbf{x}) = arg\min_{j}[D(\mathbf{x},\mathbf{c}_j)], j \in \Phi, \tag{2.2}$$

where Φ denotes all the nodes of the map and $D(\mathbf{x},\mathbf{c}_j)$ is the Euclidean distance between $\mathbf{x}$ and the weight vector $\mathbf{c}_j$.

The result of this update is to move the weight vector of the winning neuron closer to the input pattern. This updating step is given by Equation (2.3), where e is the learning rate of the network (responsible for expressing how far the node will be moved towards the input pattern):

$$\mathbf{c}_j = \mathbf{c}_j + e(\mathbf{x} - \mathbf{c}_j). \tag{2.3}$$

The weight vectors of the neighbors of the winner node $i(\mathbf{x})$ in the map are also updated towards the input pattern, but to a lesser extent, following Equation (2.5), where $h(j,i)$ is a function responsible for measuring how close in the map are the nodes j and i. Generally, a radial basis function centered at the winning node is used for the $h(j,i)$ function, as per (2.4). Remember that the radial basis function decays with the winner's discrete distance in the output map, so the nodes closer to the winner are updated with a higher factor.

$$h_{j,i(\mathbf{x})}(n) = exp\left(-\frac{\left\|r_j - r_i\right\|^2}{2\sigma^2(n)}\right), j \in \Phi, n = 0,1,2,..., \tag{2.4}$$

where r_i and r_j defines the position of the winner i and its neighbor j in the grid, Φ denotes the set of all the nodes of the map, and $\sigma(n)$ represents the width or radius of the topological neighborhood function. The $\sigma(n)$ function measures the degree to which excited nodes in the neighborhood participate in the learning process. It starts with an initially defined parameter and also decreases during the training following an exponential decay function.

$$\mathbf{c}_j = \mathbf{c}_j + h(j,i)e(\mathbf{x} - \mathbf{c}_j), \tag{2.5}$$

where i is equal to j the $h(j,i)$ is equal to 1 and the Equation (2.5) becomes the Equation (2.3). During several iterations in the training phase, the weight vectors of the network converge to model the distribution of the training dataset as best as possible. Then, in the mapping phase, each input is assigned to the prototype with the closest weight vector. When a SOM adapts to the distribution of input samples, it stretches to cover the input space, as illustrated in Figure 6.

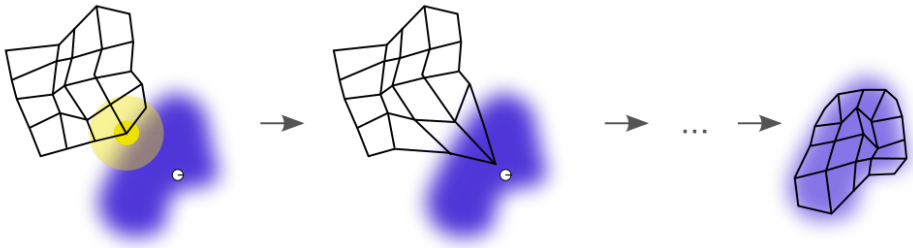


Figure 6: SOM grid during the training process adapts to the input space. The input pattern space is illustrated in blue, the current pattern is in white, the winner node or neuron is in yellow, and the black grid indicates the neighborhood. Source: Wikipedia (available under Creative Commons license)

2.3 METHODS BASED ON SOM WITH DYNAMIC TOPOLOGY AND RELEVANCE LEARNING

As *self-organizing map*, as applicable in the 2.2 section, it has some limitations that have been addressed by more recent work. Amongst them:

1. Do not work well for subspace clustering when each grouping needs to be able to take into account more strongly a different subset of the dimensions of the input patterns that can be assigned to more than one group.

2. Defining the number of nodes and the topology requires prior knowledge about the dataset so that an adequate grouping can be done.

The *dimension selective self-organizing map* (DSSOM) (Bassani & Araújo, 2012) has been proposed to deal with the first problem, and the second one has already been addressed by other methods in the literature such as Fritzke (1994, 1995); Marsland *et al.* (2002), and these methods inspired the expansion of *dimension selective self-organizing map* (DSSOM), which is called *local adaptive receptive field dimension selective self-organizing map* (LARFDS-SOM) (Bassani & Araujo, 2014). DSSOM was proposed initially for subspace clustering (Section 2.1), which allows each node to assign different relevance to each dimension to group the patterns properly. These different levels of relevances are learned throughout the training, as well as the weight vectors.

2.3.1 DSSOM

When a competition to determine the winning node occurs in SOM, each dimension of the input space is equally important. However, in *dimension selective self-organizing map* (DSSOM), each node can assign a different relevance to each dimension while calculating the distance. Specifically, given a $\mathbf{x}$ pattern, the weighted distance between the j weight vector and

the $\mathbf{x}$ pattern is given by the Equation (2.6), where ω_{ji} is the relevance given by the node j to the dimension i and m is the number of dimensions of the input data:

$$D_\omega(\mathbf{x}, \mathbf{w_j}) = \sqrt{\sum_{i=1}^{m} \omega_{ji}^2 (x_i - c_{ji})^2}. \tag{2.6}$$

The node activation is calculated by Equation (2.7), where ε is a very small value to avoid division by zero and the numerator is the sum of the components $\boldsymbol{\omega}_j$ of node j, the winner node of the competition is the one with the highest activation:

$$ac(D_\omega(\mathbf{x}, \mathbf{w}_j), \boldsymbol{\omega}_j) = \frac{\sum_{i=0}^{m} \omega_{ji}}{\sum_{i=0}^{m} \omega_{ji} + D_\omega(\mathbf{x}, \mathbf{c}_j) + \varepsilon}. \tag{2.7}$$

During network training, for each prototype j, a moving average δ_{ji} of the distance in the dimension i is computed between the weight vector $\mathbf{c}_j$ and the patterns grouped by j. Then, $\boldsymbol{\omega}_j$ is calculated as a function of $\boldsymbol{\delta}_j$, where each value ω_{ji} is inversely proportional to the variance in the i dimension of the patterns grouped by the prototype j. During the training or final grouping phases, each chosen pattern is presented to the network several times, and a different winner is calculated each time to update the DSSOM map (Bassani & Araújo, 2012). Despite being effective for subspace clustering, its training is difficult to parameterize, and its fixed topology makes it difficult to adapt the result to the dataset.

2.3.2 LARFDSSOM

In SOM and DSSOM, the number of nodes and the structure formed by their neighborhood connections are fixed. For this reason, an improvement was proposed, *local adaptive receptive field dimension selective self-organizing map* (LARFDSSOM), nodes are inserted and removed during the training, and the connections between them are updated periodically. In relation to DSSOM, LARFDSSOM has reduced computational cost, simplified parameterization, and better grouping quality. Each j node, as in the DSSOM, is associated with three vectors of the same size of the input space, in addition to a competition winner counter ($wins_j$) which is used to choose which nodes to remove:

1. Weight vector $\mathbf{c}_j$.
2. Relevance vector $\boldsymbol{\omega}_j$ [2].
3. Distance vector $\boldsymbol{\delta}_j$ (used to calculate the relevance vector).

The training of LARFDSSOM has two main phases: organization and convergence. The ANN starts with just one node, and an input pattern is chosen randomly to initialize its weight

[2] Note that relevances have more connection with the weights of the MLPs synapses than the centers of the nodes. That is why the ω, similar to w, was used.

vector. The distance vector is initialized with zeros and the relevance vector with ones. In the organization phase, there are $(t_{max} * n)$ competitions, where n is the number of input patterns, and t_{max} is an input parameter of the algorithm. In each competition, as in the previous SOM, a $\mathbf{x}$ input pattern is chosen at random and presented to the ANN. The distance between $\mathbf{x}$ and the node in LARFDSSOM is calculated, such as DSSOM, and follows Equation (2.6). The activation of each j node is given by Equation (2.7).

When there is a winner, it has its weight vector and neighbors weight vectors updated, just like a common SOM competition, with two different learning rates values for the winners and others following the Equation (2.3). Where, e is the learning rate and satisfies Equation (2.8):

$$e = \begin{cases} e_b, \text{ If it is a winner neuron;} \\ e_n, \text{ otherwise,} \end{cases} \tag{2.8}$$

being $0 < e_n < e_b < 1$. Then, the distance vectors are updated according to Equation (2.9), where β is a parameter that regulates the speed of change of the moving average:

$$\boldsymbol{\delta}_j = (1 - e\beta)\boldsymbol{\delta}_j + e\beta(\left|\mathbf{x} - \mathbf{w}_j\right|) - \boldsymbol{\delta}_j). \tag{2.9}$$

Finally, the relevance vectors are updated, using the Equation (2.10):

$$\omega_{ji} = \begin{cases} \frac{1}{1+exp(\frac{\delta_{ji}-\delta_{jimean}}{s(\delta_{jimax}-\delta_{jimin})})}, & \text{If } \delta_{jimin} \neq \delta_{jimax}; \\ 1, & \text{otherwise,} \end{cases} \tag{2.10}$$

where, δ_{jimin} is the minimum distance in the distance vector $\boldsymbol{\delta}_j$, δ_{jimax} is the maximum distance, δ_{jimean} is the average of the distances. The s is a parameter to control the smoothness of the relevance vector.

In LARFDSSOM, an important difference concerning previously mentioned algorithms is that, if the activation of the winner node in a competition does not reach a minimum value given by the parameter a_t, the winner node and its neighbors are not updated. Instead, a new node is created, with its weight vector equal to the input pattern, and its distance vector is filled with zeros and the relevance vector with ones. It is then connected to existing nodes and inserted into the map. However, if the number of nodes is at the maximum, defined by the parameter n_{max}, no node is created.

Each node j maintains its $wins_i$ counter for how many competitions it has won. For every max_{comp} competitions, all nodes with less than $(l_p * max_{comp})$ wins are removed from the map, where l_p and max_{comp} are parameters of the algorithm to control the minimum size of groups and the frequency of resets, respectively. Note that the max_{comp} parameter is multiplied by the number of patterns in the input before being used in the algorithm. After the removals, the connections of the remaining nodes are updated and their counters are reset to zero. When a node is created, it receives $(l_p * wins)$ wins, where $wins$ is the number of competitions since the

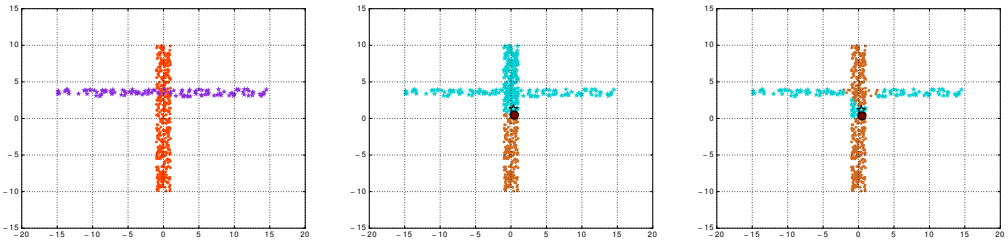
last reset, to avoid being prematurely removed.

When a node connection is created or updated, it is connected to nodes that give similar importance to the input dimensions. Specifically, two nodes i and j are connected if Equation (2.11) is satisfied, m being the number of input dimensions and p_{conn} parameter is used to regulate connectivity:

$$\left\|\boldsymbol{c}_i - \boldsymbol{c}_j\right\| < p_{conn}\sqrt{m}. \tag{2.11}$$

After the organization phase, proceeds the convergence phase, which serves to give a final adjustment to the remaining nodes. The n_{max} parameter is updated with n, the current number of nodes, and more max_{comp} competitions are held, until a final removal phase. Finally, to define which patterns belong to which groups, for each pattern, the activation of nodes is calculated, and it is assigned to every node with activation above the threshold a_t. If instead of subspace clustering, the algorithm is configured to perform the projective clustering, in which every pattern can belong to only one group, then each pattern is associated only with the node with the highest activation.

It is essential to mention that classical SOM can not accomplish subspace clustering tasks because, during its train, all dimensions are considered to have the same weight. Experiments with DSSOM (Bassani & Araújo, 2012) and LARFDSSOM (Bassani & Araujo, 2014), show that relevance learning works better on subspace clustering because it can learn the relevance (weight) of each dimension during its train, these algorithms perform better than k-means in subspace clustering benchmarks. Figure 7 illustrates a qualitative clustering results comparison of SOM and SOM with relevance learning on a toy problem.



(a) (b) (c)

Figure 7: Cluster results obtained by (b) SOM and (c) LARFDSSOM for the dataset shown in Figure (a). The centroids are represented by larger symbols. Centroids in both SOM and LARFDSSOM are almost at the same location, but LARFDSSOM better fits among the intersection of the two dimensions on the input data because of relevance learning. Source: Bassani & Araujo (2014).

2.4 REPRESENTATION LEARNING

The pioneering work in visual attention called feature integration theory (Treisman & Gelade, 1980) was proposed in psychology, in which the objective was to explain human perception and cognition. The theory defends the idea that the feature perception comes before the perception of the object. The features are recorded in the visual field, and then the brain processes these basic features, later it helps the visual attention to form the concept of what would be an object. Different visual features, such as color, orientation, spatial frequency, brightness and direction, are recorded on feature maps. The combination of feature maps forms a representation of a possible object (Begum & Karray, 2011). The feature integration theory is an acceptable model of how human visual cortex works. It begins with representation learning of simple features combined in different levels of complexity until arriving at a more elaborate abstraction, as parts of the object or even the complete object.

Many tasks that depend on data can become harder or easier, depending on how they are represented. For example, representing a circle or curve in an image could make detecting a soccer ball easier than looking at each image pixel. A good representation is the one that is capable of making the subsequent task easier (Goodfellow *et al.*, 2016). In computer vision, the algorithms can extract low-level features such as borders with filters, and then use transformations in the images to detect more common patterns. A traditional way is to use pixel and transform it into more abstract representations such as the presence of borders, detection of specific shapes, identification of categories. In the end, all this information must be grouped in order to understand the scenario (Bengio *et al.*, 2009).

As seen, some human cells perceive more straightforward information from raw input, and then, the perception of objects or parts of them by cells that perceive more complex information following a hierarchy as proposed by feature integration theory. This idea inspired deep learning techniques, which represents with a composition of layers, several hierarchical transformations of the input regardless of the type of the input, be it: sound, image or another variety of signal. Figure 8, shows the work developed by Zeiler & Fergus (2014) to represent low, medium, and high-level features with DNN.

2.4.1 Supervised Representation Learning

The representation can be learned in a supervised way, in which each input needs a respective label to guide the training process. The trained model can be used to extract representations. When ANN are composed of many layers, these representations have different levels of abstractions. If it is closer to the supervised process classifier, it tends to be more abstract and closer to a representation useful to the aim of the supervised loss function. The major drawback of supervised representation learning consist is that Labels are essential for this kind of learning process, but it is expensive to acquire useful labels to solve real-world problems.

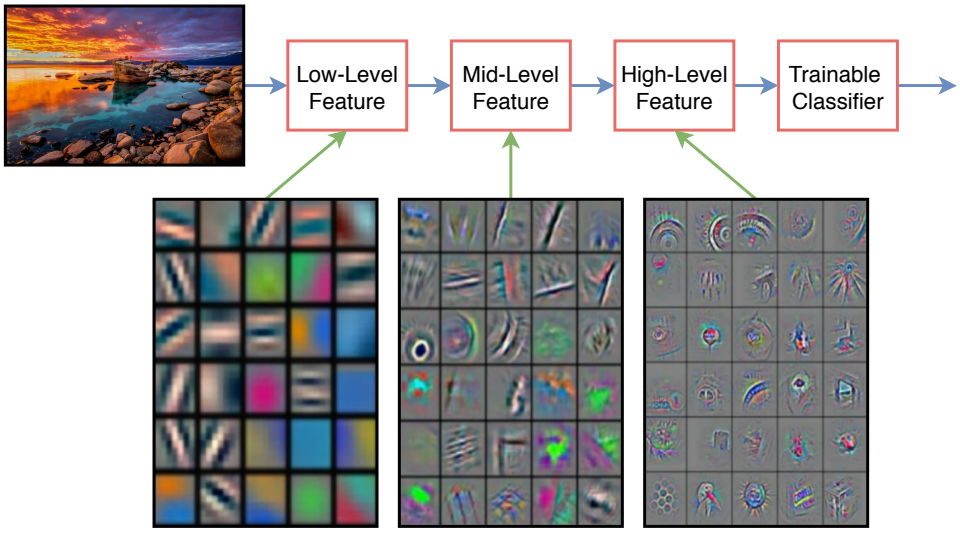


Figure 8: Feature Visualization of DNN - Low, Medium and High Level Features. Source: Adapted from Zeiler & Fergus (2014).

2.4.1.1 *Multilayer Perceptron (MLP)*

A *fully-connected network* (FCN), also known as *multilayer perceptron* (MLP), consists of multiple layers of artificial neurons, each artificial neuron is connected, and every connection has its weight. The MLP, after its training, can be used to extract features of its input and use its representation for other tasks. The MLP, illustrated in Figure 9, which receives the values (x_i) that will be used as input by the neuron, with their weights (w_i), pass through a hidden layer with the neurons h_i and its respective weights w_i^1, which pass through an activation function, and are added generating the output y. The output obtained (y) compared to the desired output (t) generates an error that is useful to train the ANN.

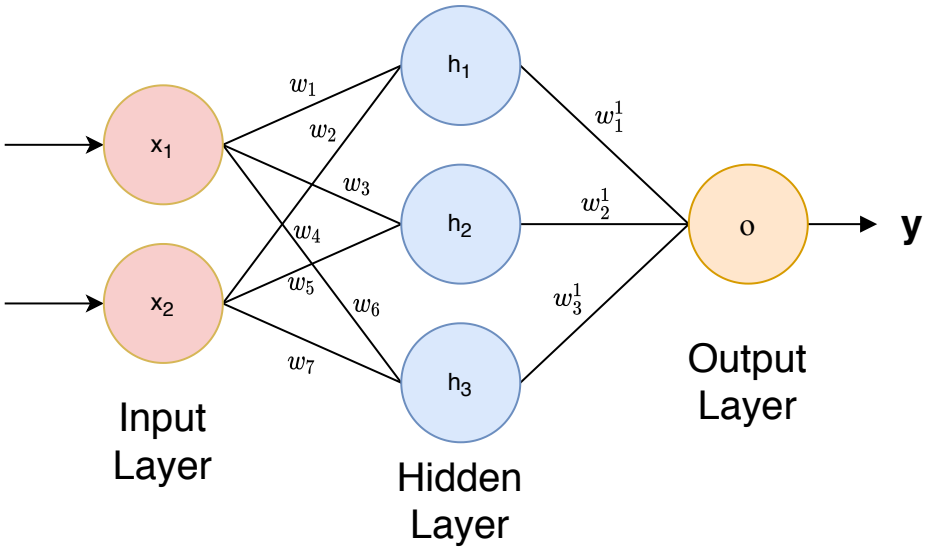


Figure 9: Multilayer Perceptron (MLP): A class of feedforward artificial neural network. x_i are the input features, with its weights (w_i). The h_i are the hidden neurons, and o is the output unit.

The training of ANN can be performed by backpropagation, which is based on the propagation of errors of layers for previous layers to carry out its weight adjustments. With backpropagation, the network can adjust its parameters to generate an output with less error. Here is a description of how to calculate the error to adjust the weights of the ANN in training phase:

a) Forward Pass: each layer calculate the sum of its inputs multiplied by its owns weight, followed by an activation function ϕ, represented in Equation (2.12) to train a network, which y is the current system output, (w_i is the weight of the neuron connection, x_i is the current value of the neuron input, n is the number of nodes in the network:

$$y = \phi\left(\sum_{i=1}^{n} w_i x_i\right). \tag{2.12}$$

b) The error E generated by the output is calculated through the error between the current output y and the desired output t, as shown by Equation (2.13):

$$E = \frac{1}{2}(t-y)^2. \tag{2.13}$$

c) Backward Pass propagates the error over the network, in which the value for updating each weight is given by Equation (2.14), which is called *stochastic gradient descent* (SGD) proposed by Nemirovski & Yudin (1978):

$$w_i = w_i - \eta \frac{\partial E}{\partial w_i}, \tag{2.14}$$

where η is the learning rate and $\frac{\partial E}{\partial w_i}$ is the partial derivative of the error in relation to each neuron weights. The SGD is used to optimize objectives estimating the parameters of the ANN.

2.4.1.2 *Supervised Deep Learning*

Supervised Deep Learning techniques can be used for representation learning. After its training, the deep model can be used to extract features of the input and these features can be used in other tasks such as clustering. DL consists of several amounts of layers. No one can deny that *deep learning* (DL) changed the paradigm of classical *machine learning* (ML) algorithms. Until the last decade, ML algorithms relied on feature engineering, in which a feature engineer was responsible for selecting the best features for the aim problem. Features are transformations on the input data that helps the algorithm solve a problem. DL techniques aim at finding adequate features representations automatically by training a *deep neural networks* (DNN). Deng & Yu (2014) suggest several definitions of DL. The following list presents some of them:

1. Class of ML techniques for exploring layers of nonlinear information by processing and extracting supervised and unsupervised features, as well as performing transformations to analyze and classify patterns.

2. ML subfield that uses algorithms to represent levels to model complex relationships of the data. Which high-level features are defined in terms of low-level features.

3. A subset of ML methods based on representation learning. An observation (ex: an image), can be represented in several ways (ex: a set of pixels), but some representations are easier to learn specific tasks.

4. Set of algorithms in ML that aim to learn multiple levels, corresponding to different levels of abstraction. The levels in learned models correspond to different levels of concepts, where the highest order concepts are defined through lower-order concepts, and the low-order concepts can help define several of the high-order concepts.

Therefore, the main goal of a DNN is to create a representation of the data that is useful for solving complex problems. The success of DL remains on its supervised part. The unsupervised still needs to be improved to reach good results as compared to supervised.

2.4.2 Unsupervised Representation Learning

In the early days of DL, deep representations researches were composed of unsupervised learning techniques, e.g. training RBM (Hinton, 2012), or AE based models (Vincent *et al.*, 2008), like *stacked autoencoder* (SAE). Generally, SAE were trained in a layerwise fashion using the backpropagation algorithm with SGD.

Unsupervised learning as a pretraining stage was usual in DL, such as in Hinton *et al.* (2006). This learning process is known to produce better DL model initialization (Bengio *et al.*, 2007) without the worry of acquiring labels. Unsupervised pretraining guides the optimization towards basins of attractions of minima that allow better generalization from training and add robustness to a deep architecture (Erhan *et al.*, 2010).

The DL high-level representations can be generally extracted from one layer of ANN, for example, from *autoencoders* (AE) (Vincent *et al.*, 2008) or a *restricted boltzmann machine* (RBM) (Hinton, 2012), or from CNN layers (Saito *et al.*, 2017). AE are in fact fundamentally connected to clustering as shown by Baldi (2012). In the following Section 2.4.2.1 explains in details about *autoencoders* (AE) and the importance of the latent space generate by its training.

2.4.2.1 Autoencoders

Autoencoders (AE) are a kind of an auto-associative ANN that is used for dimensionality reduction and feature learning. During training, the AE aims to minimize the reconstruction error, i.e., the error between its input and the respective output so that it produces an output

close to the input. The training process of classic AE does not need labels, only the input itself, and for this reason, it is considered an unsupervised data-driven process. The hidden layer, **h**, in the middle is usually constrained to be a narrow bottleneck producing a latent space. The system can minimize the reconstruction error by ensuring the hidden units capture the most relevant aspects of the data (Murphy, 2012). The AE network consists of two parts: an encoder function $f(\mathbf{x})$, where $\mathbf{x}$ is the input data, that maps the input to the latent space and a decoder that produces a reconstruction $g(f(\mathbf{x})) = \hat{\mathbf{x}}$, where $\hat{\mathbf{x}}$ is the reconstructed input data. If the AE converges adequately, it learns the map $g(f(\mathbf{x})) = \mathbf{x}$. The AE learning process is described as minimizing Equation (2.15), which can be trained with SGD:

$$\mathscr{L}(\mathbf{x}, g(f(\mathbf{x}))) = \left\|\mathbf{x} - g(f(\mathbf{x}))\right\|^2, \mathbf{x} \in \Phi, \tag{2.15}$$

where Φ is the set of samples in the training dataset. When the AE has a constraint where **h** dimension is smaller than the input **x** dimension, it is called undercomplete AE. This constrain forces the AE to learn the more important and useful features of the input to encode it. A deep AE can learn more powerful nonlinear generalization compared to *principal component analysis* (PCA) (Goodfellow *et al.*, 2016). Figure 10 illustrates an example of undercomplete AE, where the input is retracted in the encoding layer and expanded in the decoding layer.

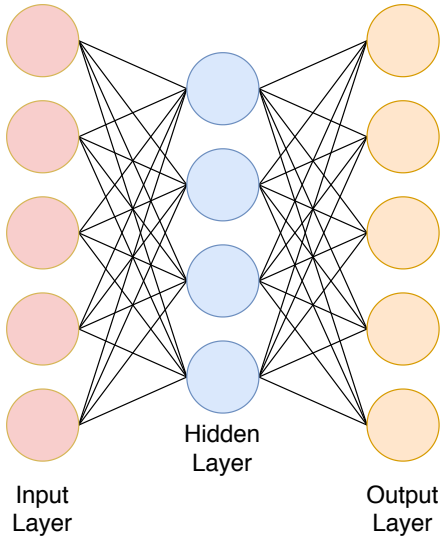


Figure 10: A simple example of undercomplete AE. The neurons in red represent the input layer, the neurons in blue represent the hidden layer, and the neurons in orange represent the output layer.

3 REVIEW OF THE LITERATURE

Deep Learning (DL) learns how to map one vector to another, given a large amount of labeled data. The success of DL relies on solving supervised learning problems, which labels are needed. Several DL techniques have been proposed to tackle unsupervised learning problems, but none have truly solved it in the same way as the supervised counterpart. The combination of clustering with deep learning is a family of approaches aiming at unsupervised learning tasks. These techniques are being applied to obtain richer deep representations of the data, and at the same time, it creates clusters in an unsupervised fashion way by a data-driven process.

The purpose of this chapter is to introduce some concepts closer to this thesis. Section 3.1 starts with the deep clustering, the next Section 3.2 explains the essential building blocks for deep clustering, Section 3.3 details the joint loss function for deep clustering, and end Section 3.4 detail the state-of-art deep clustering self-organizing maps baselines.

3.1 DEEP CLUSTERING

The DC research field resulted from the application of DL methods in clustering problems. In Figure 11, the taxonomy of methods proposed by Nutakki *et al.* (2019) and adopted in the rest of this work is illustrated. The DC scenario differs from conventional approaches varying their algorithm structure, network architectures, loss functions, and optimizations methods for training. In the following sections, some core concepts will be discussed to introduce the reader to the DC scenario.

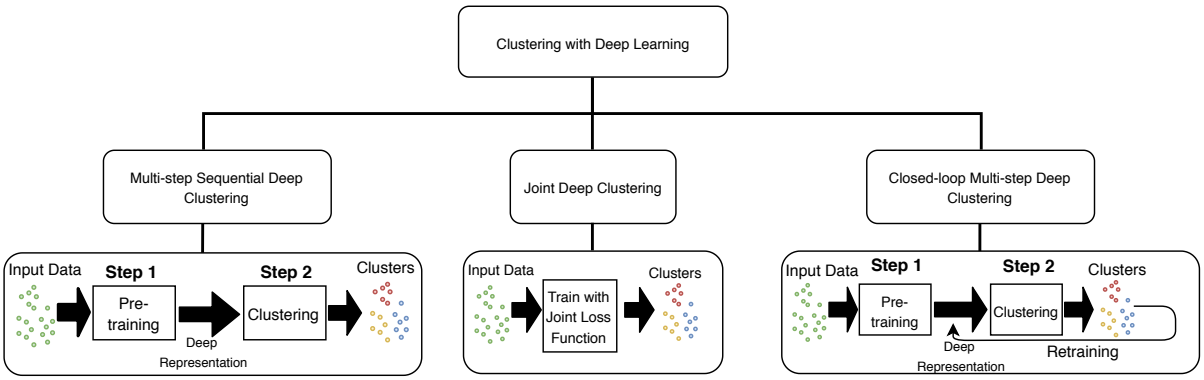


Figure 11: Deep Clustering taxonomy proposed by Nutakki *et al.* (2019). It divides the field into three main sub-fields differing in the number of steps and the presence of feedback from the clustering step. Source: Adapted from Nutakki *et al.* (2019).

The Nutakki *et al.* (2019) taxonomy's divides clustering with deep learning into three main topics listed below:

- *multi-step sequential deep clustering* (MSSDC)
- *closed-loop multi-step deep clustering* (CLMSDC)

- *joint deep clustering* (JDC)

3.1.1 Multi-Step Sequential Deep Clustering

The MSSDC is a forward process divided into two main steps, as illustrated in Figure 12. The first one learns a richer representation of the input data using DL techniques, e.g., CNNs or AE. The subsequent step performs clustering on that deep representation (or latent representation) learned by the previous step. An example of this family of methods is Fast Spectral Clustering (Banijamali & Ghodsi, 2017), where in the first step, an AE is trained to produce a latent or deep representation, and in the second step, its representation is used as input to clustering with k-means.

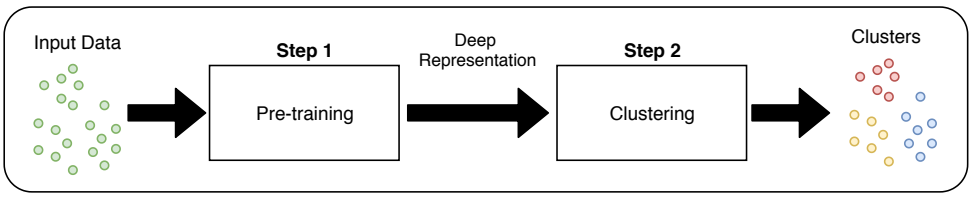


Figure 12: Multi-Step Sequential Deep Clustering: A forward Deep Clustering approach composed of two steps: First step for representation learning, and the second for clustering. Source: Adapted from Nutakki *et al.* (2019).

Generally, MSSDC is the first DC technique experimented when clustering with DL algorithm is implemented because it is easier to evaluate the deep learning or clustering algorithms as isolated building blocks. After the evaluation of each block, the one responsible for representation learning and the other module responsible for the clustering task, the problem is minimized to discover the best representation that suits the specific clustering technique, where the dimension of representation matches the clustering algorithm input shape.

In this kind of DC technique, success is related to the representation created in the first step. Consequently, a good strategy to have better results is evaluating the representation produced before applying the clustering technique. A simple qualitative analysis consists of visualizing the representation or how to disentangle its dimensions as performed by Li *et al.* (2018). If a dimension is disentangled, its features can generally be isolated, helping the clustering process, Figure 13 illustrates an example of the disentangled feature.

Figure 13: Latent features learned in the Flower dataset. The pictures are generated by varying a latent variable, while others are zero. The color temperature of the flower is an example of a disentangle feature. Source: Li *et al.* (2018).

In MSSDC is important to mention that each block of the technique is easy to maintain and test separated due to its two steps. The principal drawback is the need to run two separate tasks to perform the whole algorithm demanding a larger running time.

3.1.2 Closed-Loop Multi-step Deep Clustering

The CLMSDC is a family of DC methods similar to MSSDC, however, the clusters can be used to adjust the deep representation, as illustrated in Figure 14. Therefore, the algorithm performs one step of, for example, an AE and another step of the clustering method during its training. An example of this method is the *deep embedded clustering* (DEC) that uses an AE as network architecture and k-means for clustering, while training. The algorithm performs one step of the AE, and one step of the k-means and the AE is adjusted with the cluster loss by backpropagation.

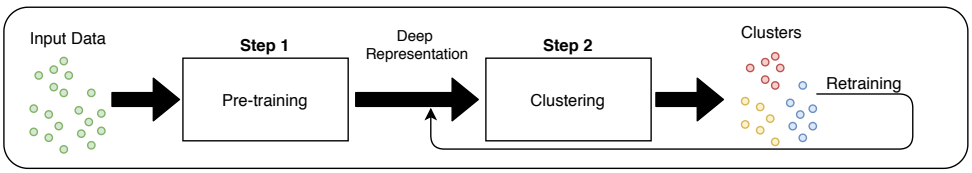


Figure 14: Closed-Loop Multi-step Deep Clustering: A Deep Clustering approach composed of two steps, one for representation learning and the second for clustering. The clusters are used to adjust the feature representation performing a loop. Source: Adapted from Nutakki *et al.* (2019).

Recently, Caron *et al.* (2018) proposed the DeepCluster. It is a CLMSDC model that provides a technique to perform DC using the k-means cluster algorithm to create pseudo-labels and adjust the weights of a deep neural network. The DeepCluster illustrated in Figure 15, shows that good representations of visual features can be learned with CLMSDC approach.

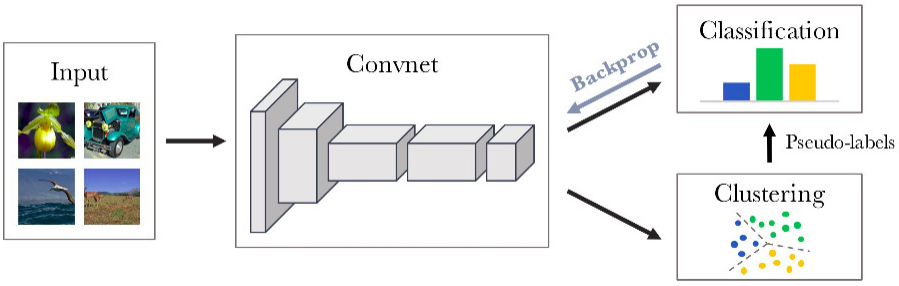


Figure 15: DeepCluster: A clustering method that jointly learns the parameters of a neural network and the cluster assignments of the resulting features. Source: Caron *et al.* (2018).

In CLMSDC is important to mention that this technique has the same positive points of the aforementioned MSSDC method, but it has the advantage of using its clusters to fine-tuning the representation. The principal drawback involves tuning the clusters the right way to avoid a significant perturbation that makes the whole system unstable, e.g., degenerating the clusters and making the loss function diverges rather than fine-tuning it.

3.1.3 Joint Deep Clustering

Differently from the methods above of DC that can be accomplished with two main steps performed in a sequential way or iteratively, the JDC is performed in only one step, in which the representation of the data is learned with a combined or joint loss function responsible for a good representation learning while also performs clustering, as illustrated by Figure 16.

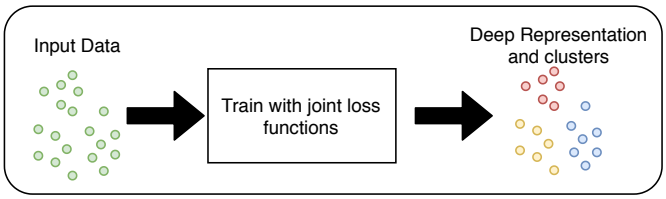


Figure 16: Joint Deep Clustering: Deep Clustering approach that learns representation and clustering jointly. Source: Adapted from Nutakki *et al.* (2019).

In this scenario, FaceNet proposed by Schroff *et al.* (2015), illustrated in Figure 17, combines a CNN and a Triplet Loss (a loss that receives positive and negative samples to approximate and separate objects in the clusters in the space) to perform tasks such as face recognition and clustering jointly.

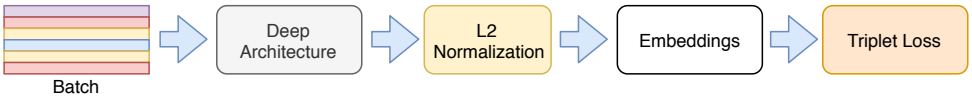


Figure 17: FaceNet: Combines a CNN and a Triplet Loss to perform tasks such as face recognition and clustering jointly. Source: Adapted from Schroff *et al.* (2015).

In JDC is important to mention that due to the combined loss function, it tends to learn better representations when the loss function is fine-tuned, and it only requires one step to perform the algorithm. The main drawback consists of debugging errors that can exist in the whole process, due to the combination of each loss function term, it is harder to know which model is damaging more the learning process.

3.2 ESSENTIAL BUILDING BLOCKS FOR DEEP CLUSTERING

The development of a framework to achieve DC needs some suitable crucial components, such as useful representations and a robust cluster algorithm, or a loss function that captures the essential points of these representations to perform clustering. The present section briefly introduces a set of tools commonly used for unsupervised learning and to build DC models

3.2.1 Artificial Neural Network Architecture for Deep Clustering

The ANN are essential to transform the input data into a new feature representation useful to DC. The following list details the work done by Aljalbout *et al.* (2018) and Min *et al.* (2018) to address some of these architectures that are useful for DC:

- Convolutional Neural Network (CNN): (LeCun *et al.*, 1998) is one of the most powerful and applied tools to run with complex high-dimensional data (e.g., image and sound). The idea behind the CNN is the ability to, during its train, learn the best convolution filters (kernels, or weights) to solve a task based on its loss function. The CNN can be trained with a clustering loss function directly without a pretraining step, although a good initialization would significantly boost the performance (Min *et al.*, 2018).

- Deep Belief Network (DBN): *deep belief networks* (DBN) are generative graphical models composed of several unsupervised networks such as RBM (Hinton, 2009) or AE (Bengio *et al.*, 2007). Each unsupervised network can be seen as a hidden layer that serves as a visible layer for the next one. Those layers are named latent variables or hidden units. This class of DNN holds connections between the layers, but not between units inside each layer. In DBN, the unsupervised networks are responsible for learning to extract a deep representation of the input data. When trained in an unsupervised fashion strategy, it can be employed to reconstruct its input, and the layers then learn to detect features. Commonly, after the unsupervised training, a fine-tuned process makes the neural network adjusted to perform classification.

- Variational Autoencoder (VAE): *variational autoencoders* (VAE) is a class of AE able of performing sample generation. The VAE determines the data distribution making its encoder learns, a vector of means, and a vector of standard deviation, using

Bayesian inference (Kingma & Welling, 2013; Rezende *et al.*, 2014). The Bayesian inference is a statistical inference method in which Bayes' theorem is applied to update probabilities of the model.

The list mentioned above is a useful source of architectures for developing a DC framework. To pick these architectures, many aspects of the ANN need to be considered and adjusted with the research goal. In this work, the AE was chosen as a start point because the latent space features have been used for clustering tasks, but the others could also be used as the base ANN architecture.

3.2.2 Deep Clustering Loss Functions

The purpose of the loss function is to guide the training process towards its objective. A DC loss function leads the ANN to learn clustering-friendly representations. In DC, Min *et al.* (2018) categorize the loss functions in two sets: the clustering loss and the non-clustering loss. The clustering loss is responsible for the cluster centroids and cluster assignments of samples, within this loss, the neural network can obtain the clusters directly (e.g., k-means loss (Yang *et al.*, 2017), cluster assignment hardening loss (Xie *et al.*, 2016), agglomerative clustering loss (Beeferman & Berger, 2000)). The non-clustering loss makes small adjustments in the neural network, guiding to learn more interesting representations for clustering (e.g., locality-preserving loss (Huang *et al.*, 2014), group sparsity loss (Huang *et al.*, 2014), sparse subspace clustering loss (Peng *et al.*, 2017)).

Nutakki *et al.* (2019) describe the four most important types of losses that can be used in DC. The first one comes purely from learning a deep representation using deep learning techniques, and does not depend on any clustering quality measure (e.g., the reconstruction error of an autoencoder or matrix factorization). The second is similar to the principal clustering loss of Min *et al.* (2018), which comes from a clustering process. The third type is a joint loss function that combines clustering loss and the non-clustering with a hyperparameter. The fourth is a loss function to obtain harder cluster partitions in case of soft clustering. The soft clustering allows that an input belongs to multiple clusters.

Here, the main type of loss consider is joint loss function. The joint loss function can be divided into two main parts: non-clustering loss and clustering loss. The first, one responsible for the representation learning (e.g., AE loss, ANN loss); in the case of AE, it is called reconstruction loss. The second part is responsible for creating a better clustering representation (e.g. k-Means loss, SOM loss[1]). The following list detail some of these losses:

- ANN Loss: This is the standard scenario of a deep learning model, in which the model is trained as usual with its loss function. After the training, the model weights can be used to extract features, and these features subsequently can be used in clustering algorithms.

[1]The SOM loss is described in *deep embedded self-organizing map* (DE-SOM) algorithm.

- AE Reconstruction Loss: The reconstructions loss of an AE, already described in (2.15), can be used for learning representations in the latent space, extracting features that can be used for clustering.

- k-Means Loss: This loss ensures that the new representation is k-means-friendly (Yang *et al.*, 2017), i.e., the input data are distributed around the cluster centers. To accomplish this task, an ANN is trained with the following loss function:

$$\mathscr{L}(\theta) = \sum_{i=1}^{N} \sum_{k=1}^{K} \|z_i - \mu_k\|^2, \tag{3.1}$$

where z_i is an embedded data point, μ_k is a cluster center, N is the number of embedded data points, and K is the number of clusters. The goal of minimizing this loss function is to make the distance between the data point and its assigned cluster small.

3.3 DEFINING JOINT LOSS FUNCTIONS

In the case of the joint loss function, which a clustering and a non-clustering loss function are used, this work is going to follow the definition of Aljalbout *et al.* (2018) to combine the two different losses, as described by the following equation (3.2), which if the hyperparameter α is set to one it has only clustering loss, and if α is set to zero, it has only non-clustering loss:

$$\mathscr{L}(\theta) = \alpha \mathscr{L}_c + (1-\alpha)\mathscr{L}_n(\theta), \tag{3.2}$$

where $\mathscr{L}_c(\theta)$ is the clustering loss, $\mathscr{L}_n(\theta)$ is the non-clustering loss, and $\alpha \in [0;1]$ is a constant specifying the weighting between the two different losses contribution. This constant is an additional hyperparameter for the ANN training. This hyperparameter can be adjusted during training following some schedule, e.g., if it hits a clustering metric plateau, it can decrease or increase the parameter to adjust the contribution of the clustering loss, depending on if the metric is being maximized or minimized. The following are strategies to assign and schedule these values of α:

- Pre-training, fine-tuning: The training method is classified as a MSSDC, which first, α is set to 0, i.e., the ANN is trained using the non-clustering loss only. Next, α is set to 1, i.e., the non-clustering network branches are removed (e.g., autoencoder's decoder) the features are extracted, and the clustering loss is used to train (fine-tune) the obtained network or to cluster these features.

- Combine the different joint training losses: This training method is classified as a JDC, in which both losses contribute to the training process. The hyperparameter

α varying between 0 and 1 exclusively ($0 < \alpha < 1$), e.g $\alpha = 0.5$, i.e. the network training is affected by both loss functions equally.

- Dynamically adjust hyperparameter: The α is varied during the training depending on a chosen schedule. For instance, start with a low value for α and gradually increase it in every phase of the training, i.e., the non-clustering loss starts with the weighting of one for the training process and decays to zero, while the clustering loss counterpart increases complementary (going from zero to one), so first the ANN adjust its representation and after it adjusts the clusters.

The concepts described above are frequently adopted in DL for combining different loss components and optimize them.

3.4 DC BASELINES WITH SOM

In this section, the DE-SOM and *self organizing map variational autoencoder* (SOM-VAE) are introduced to explain some ideas that were applied to the development of our proposed SOM algorithm. The mathematical notation of each baseline method was preserved as presented in each original paper, but some interpretations and observations were added to understand these methods.

3.4.1 Deep Embedded Self-Organizing Map (DE-SOM)

The DE-SOM (Forest *et al.*, 2019) uses an AE combined with a classical bidimensional SOM grid. The DE-SOM is a JDC algorithm that jointly learns the latent space with the AE and cluster with SOM loss. Forest *et al.* (2019) claim that the AE can learn a low-dimensional representation that improve the SOM performance, while self-organization and representation learning is achieved in the same step improving the model performance (in terms of classification and training speed convergence). The principal points of the DE-SOM consist of learning a continuous latent space with the AE, and also, apply a Gaussian neighborhood with exponential radius decay to categorize each prototype. Figure 18 illustrates the DE-SOM architecture with the SOM learning to cluster in the latent space.

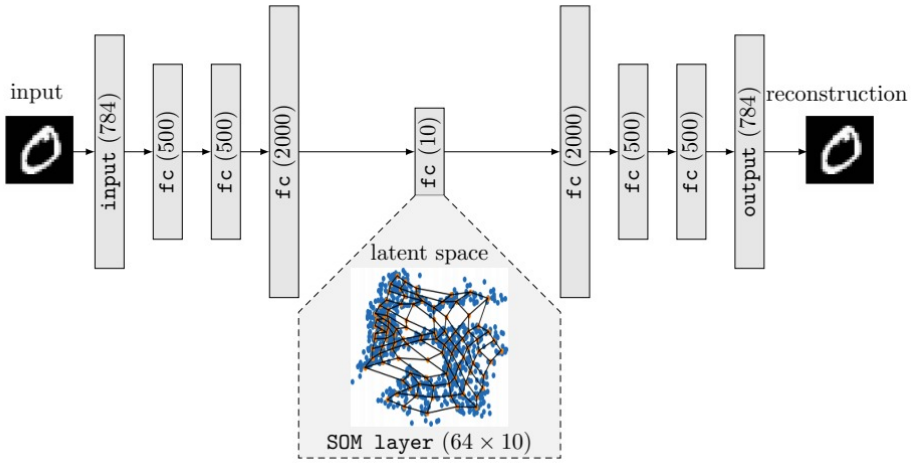


Figure 18: DE-SOM architecture with an 8×8 map, it learns 64 prototypes. The architecture consists of an autoencoder with a SOM coupled in the latent space features. Source: Forest *et al.* (2019)

The SOM contains K units representing the prototype vectors $\{\boldsymbol{m}_k\}_{1 \leqslant k \leqslant K}$. $\delta(.,.)$ is the distance between two nodes on the map. A Gaussian neighborhood function $\kappa^T(d) = e^{\frac{-d^2}{T^2}}$ was used to control neighborhood's radius, which depends on a parameter T. This parameter decays exponentially at each training iteration. $\boldsymbol{W}_e$ is the encoder and $\boldsymbol{W}_d$ the decoder parameters. The features in the latent space are represented by $\boldsymbol{z}_i = \boldsymbol{f}_{w_e}(\boldsymbol{x}_i)$, which embeds the input $\boldsymbol{x}_i$, and the decoders' reconstruction is described by $\tilde{\boldsymbol{x}}_i = \boldsymbol{g}_{W_d}(\boldsymbol{z}_i)$. The combined DE-SOM loss is composed of two parts, described by the following Equation (3.3):

$$\mathscr{L}(\boldsymbol{W}_e, \boldsymbol{W}_d, \boldsymbol{m}_1, ..., \boldsymbol{m}_K, \chi) = \mathscr{L}_r(\boldsymbol{W}_e, \boldsymbol{W}_d) + \gamma \mathscr{L}_{SOM}(\boldsymbol{W}_e, \boldsymbol{m}_1, ..., \boldsymbol{m}_K, \chi), \tag{3.3}$$

where the first term is the AE reconstruction loss ($\mathscr{L}_r$) and the second term is the SOM loss ($\mathscr{L}_{SOM}$). The coefficient γ trades off between reconstruction and SOM losses. The SOM loss depends on the parameters $\boldsymbol{m}_k$ and the assignment function $\chi(z) = argmin_k \|\boldsymbol{z} - \boldsymbol{m}_k\|^2$. The authors do not explain the hole of $\delta(.,.)$ in the loss function, but it seems to be a factor that penalizes the nodes that are distant in the map but have similar features and vice-versa. Equation (3.4) describes this loss:

$$\mathscr{L}_{SOM} = \sum_i \sum_{k=1}^{K} \kappa^T(\delta(\chi(\boldsymbol{f}_{W_e}(\boldsymbol{x}_i)), k)) \left\| \boldsymbol{f}_{W_e}(\boldsymbol{x}_i) - \boldsymbol{m}_k \right\|^2. \tag{3.4}$$

When the parameter T approaches zero, the SOM loss becomes identical to a k-means loss, as in Equation (3.5):

$$\lim_{T \to 0} \mathscr{L}_{SOM} = \sum_i \left\| \boldsymbol{f}_{W_e}(\boldsymbol{x}_i) - \boldsymbol{m}_{(\chi(\boldsymbol{f}_{W_e}(\boldsymbol{x}_i))} \right\|^2. \tag{3.5}$$

These peculiarities make the DE-SOM capable of adjusting the latent representation during the training. When the model converges, the latent space becomes more stable, allowing SOM to work more effectively. DE-SOM is one of the baseline methods that will be compared with our proposed model.

3.4.2 Self Organizing Map Variational Autoencoder (SOM-VAE)

The SOM-VAE (Fortuin *et al.*, 2018) is a JDC model that combines ideas from SOM with *vector quantised-variational autoencoder* (VQ-VAE) in a discrete latent space. The authors devised a novel framework for interpretable discrete representation learning on time series and show that the latent probabilistic model improves clustering. In Figure 19, the SOM-VAE model is illustrated.

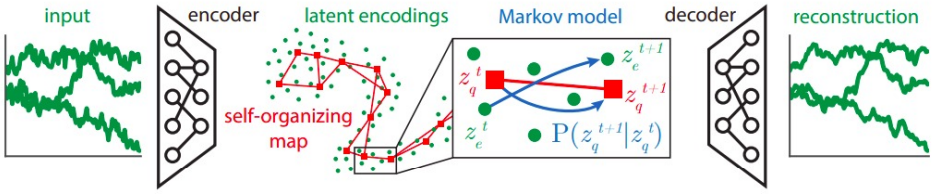


Figure 19: Schematic overview of SOM-VAE architecture. In green, the time series is the input data encoded by an ANN into the latent space. The encoded points (z_e) in the latent space is assigned to a cluster z_q by SOM, in red. A Markov transition model, in blue, is learned to predict the next discrete representation (z_q^{t+1}) given the current one (z_q^t). The discrete representations can then be used as input to the decoder to reconstruct the encoded data into the input space. Source: Fortuin *et al.* (2018).

In SOM-VAE, an input $x \in \mathbb{R}^d$ is encoded to $z_e \in \mathbb{R}^m$ by computing $z_e = f_\theta(x)$, where $f_\theta(.)$ represents the encoder. The data encoded in the latent space is then assigned to a prototype $z_q \in \mathbb{R}^m$ with the prototypes being element of the dictionary $E = \{e_1, ..., e_k | e_i \in \mathbb{R}^m\}$ by sampling $z_q \sim p(z_q|z_e)$. The distribution of z_q is modeled by a categorical distribution with probability mass 1 on the closest embedding to z_e, i.e. $p(z_q \mid z_e) = \mathbb{1}\left[z_q = \arg\min_{e \in E} \|z_e - e\|^2\right]$, where $\mathbb{1}[.]$ is 1 if $z_q \in E$ or 0 if $z_q \notin E$.

The reconstruction $\hat{x}$ of the input can then be computed as $\hat{x} = g_\phi(z)$, where $g_\phi(\cdot)$ is the output of the decoder network. Since the encodings and prototypes are in the latent space, two different reconstructions can be computed, one derived of the directly input encoding z_e and other from the cluster z_q, namely $\hat{x}_e = g_\phi(z_e)$ and $\hat{x}_q = g_\phi(z_q)$.

The SOM-VAE loss function for a input x is described by (3.6):

$$\mathscr{L}_{\text{SOM-VAE}}(x, \hat{x}_q, \hat{x}_e) = \mathscr{L}_{\text{reconstruction}}(x, \hat{x}_q, \hat{x}_e) + \alpha \mathscr{L}_{\text{commitment}}(x) + \beta \mathscr{L}_{\text{SOM}}(x), \tag{3.6}$$

where x, z_e, z_q, $\hat{x}_e$ and $\hat{x}_q$ are defined as above and α and β are weighting hyperparameters. Every term in the SOM-VAE loss function was designed to optimize a different model component. The

first term, $\mathscr{L}_{\text{reconstruction}}\left(x,\hat{x}_q,\hat{x}_e\right)$, is the VAE reconstruction term:

$$\mathscr{L}_{\text{reconstruction}}\left(x,\hat{x}_q,\hat{x}_e\right) = \left\|x-\hat{x}_q\right\|^2 + \left\|x-\hat{x}_e\right\|^2, \tag{3.7}$$

where the first subterm $\left\|x-\hat{x}_q\right\|^2$ encourages the assigned SOM node $z_q(x)$ to be an informative representation of the input and $\left\|x-\hat{x}_e\right\|^2$ encourages the encoding $z_e(x)$ to also be an informative representation. This term correspond to the *evidence lower bound* (ELBO) of the VAE (Kingma & Welling, 2013). The ELBO is optimized using the *Kullback–Leibler divergence* (KL divergence). The KL divergence, also called relative entropy is a measure of how one probability distribution is different from a second. The KL divergence of a discrete probability distributions P and Q defined on X probability space can be calculate with $D_{\text{KL}}(P\|Q) = \sum_{x\in\mathscr{X}} P(x)\log\left(\frac{P(x)}{Q(x)}\right)$.

The term $\mathscr{L}_{\text{commitment}}$ helps the encoded input and the prototype to be close to each other and is defined as $\mathscr{L}_{\text{commitment}}(x) = \left\|z_e(x)-z_q(x)\right\|^2$. In fact, the closeness of encodings and prototypes should be expected with the $\mathscr{L}_{\text{reconstruction}}$ term, but the commitment term was explicitly added to get the gradient information about z_q.

The last term is the SOM loss defined as $\mathscr{L}_{\text{SOM}}(x) = \sum_{\tilde{e}\in N\left(z_q(x)\right)} \left\|\tilde{e}-\text{sg}\left[z_e(x)\right]\right\|^2$, where $N(\cdot)$ is the collection of neighbors in the discrete space as defined above and $\text{sg}[\cdot]$ is the gradient stopping operator used to set the gradient to 0 during the backward pass. The stop operator helps the neighbors of the prototype z_q to also be close to z_e guiding the organization process of the SOM. These peculiarities make the SOM-VAE capable of adjusting the latent representation during the training. SOM-VAE is the second baseline method that is going to be used to compare with our proposed model.

4 PROPOSED MODEL

In this chapter, we are going to detail the path and the main points in the development of this work. We split it into three principal items: (i) LARFDSSOM in *multi-step sequential deep clustering* (MSSDC), (ii) SOM-RL in *multi-step sequential deep clustering* (MSSDC) and (iii) SOM-RL in *joint deep clustering* (JDC). The LARFDSSOM in MSSDC consists in bring the LARFDSSOM model to the *deep clustering* (DC) context using a supervised pre-trained model to extract features and train LARFDSSOM with these features to perform cluster. The SOM-RL in *multi-step sequential deep clustering* (MSSDC) consists on a SOM layer that we trained with unsupervised features from a pre-trained encoder. In SOM-RL in *joint deep clustering* (JDC), we propose some losses capable to train the SOM-RL in a fully unsupervised way in JDC, training jointly the autoencoder and the SOM.

The division of this work splits our contributions to each of our main objectives: (i) Use complex data such as images with SOM and DL models, (ii) develop a SOM layer capable of being trained fully unsupervised, and (iii) train the SOM layer end-to-end performing clustering and representation learning simultaneously. First, we start with the MSSDC approach to test the capabilities of LARFDSSOM algorithm with DL in clustering images. Second, we evolved the model to a layer that learns in the MSSDC scenario. Third, we define and trained a SOM layer in the JDC task. The subsequent sections explain each one of these items.

4.1 LARFDSSOM IN MSSDC

The LARFDSSOM in MSSDC scenario, exemplified in Figure 20, was the first module that we proposed in this research. We extended the work of Tuytelaars *et al.* (2010); Kinnunen *et al.* (2011) for UVOC tasks using MSSDC (Medeiros *et al.*, 2019).

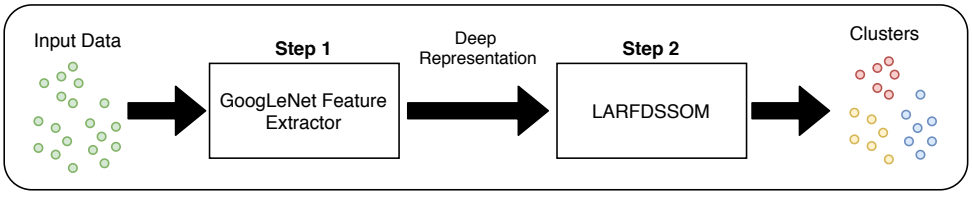


Figure 20: Proposed LARFDSSOM in the MSSDC model: In the first step, GoogLeNet's feature extractor is responsible for transform the input data into deep representation, and in the next step, the LARFDSSOM uses the deep representation input for clustering.

The UVOC task consists of learning to recognize basic entities in images typically using a bag-of-visual-features representation, Figure 21 illustrates such UVOC procedure. In Tuytelaars *et al.* (2010), interest points are detected with many classical detectors. Then, the extracted features are grouped by similarity to build a set of features clusters, i.e., an alphabet of features used after the learning process to produce image representations. Therefore, each image is

represented by its histogram of the occurrence of features throughout the features alphabet. Finally, images with similar histograms, closer enough in the feature space considering euclidean distance with a threshold, should contain similar objects; hence, they should be grouped in the same cluster.

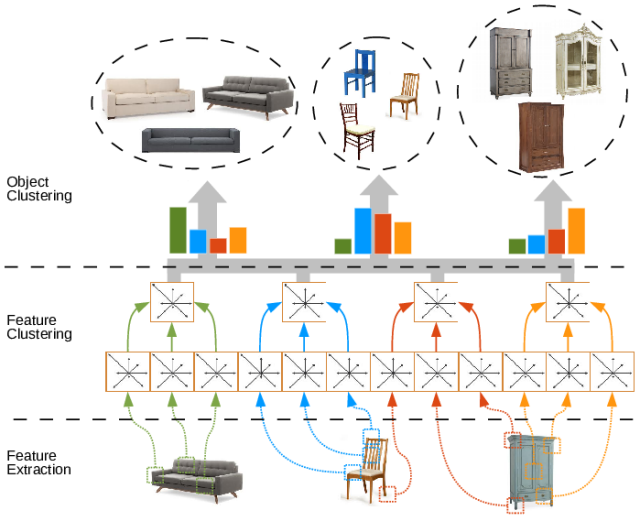


Figure 21: UVOC task using feature extraction and object clustering. First, the features are extracted, and in the following step, these features are clustered. Source: Medeiros *et al.* (2019)

During the development of our module, we selected the GoogLeNet CNN (Szegedy *et al.*, 2015) pre-trained on ImageNet (Deng *et al.*, 2009) dataset as a feature extractor. The GoogLeNet, which is shown in Figure 23, achieved the best results in classification and competition image tasks ImageNet Large Scale Visual Recognition Challenge (ILSVRC2014) (Szegedy *et al.*, 2015). The architecture is composed of several Inception modules, illustrated in Figure 22, to allow more efficient computation. These modules have convolution operation running in parallel with three different filter sizes: (1×1), (3×3), and (5×5), followed by max-pooling. The 1×1 filters reduce the dimensions of the ANN and perform faster computations when compared with fully-connected layers.

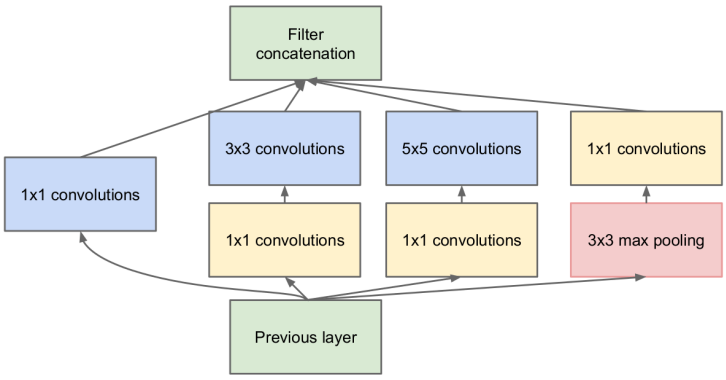


Figure 22: Inception module v1 with max-pooling. Source: Szegedy *et al.* (2015)

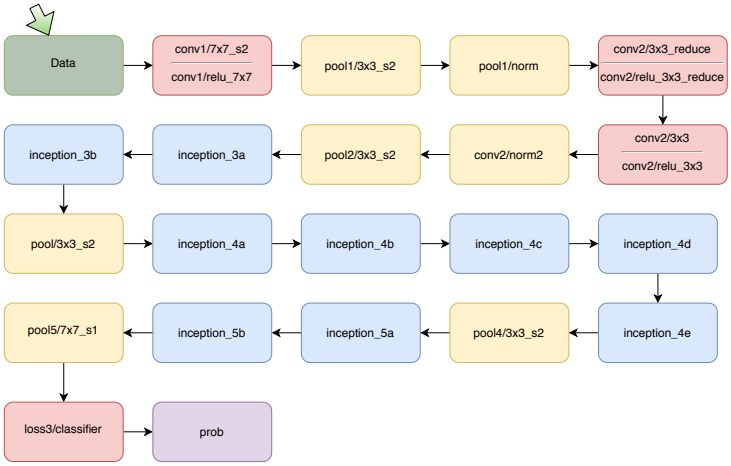


Figure 23: GoogLeNet Architecture: In green is the input data, in red the convolutions and the last classifier, in yellow the pooling and normalization layers, in blue the inception module, and in purple the probability output. Source: Adapted from (Szegedy *et al.*, 2015)

In Figure 24, the model is described, so first, we removed the GoogLeNet last classifier and extracted the features from the input dataset after we used the LARFDSSOM for clustering these features. This approach is a good starting point for developing a DC framework, remembering that after we evaluate each sub-module responsible for the feature extraction and the SOM module responsible for the clustering, we can run them together in two-steps to perform the DC task. We used the original C++ implementation of LARFDSSOM developed by Bassani & Araujo (2014).

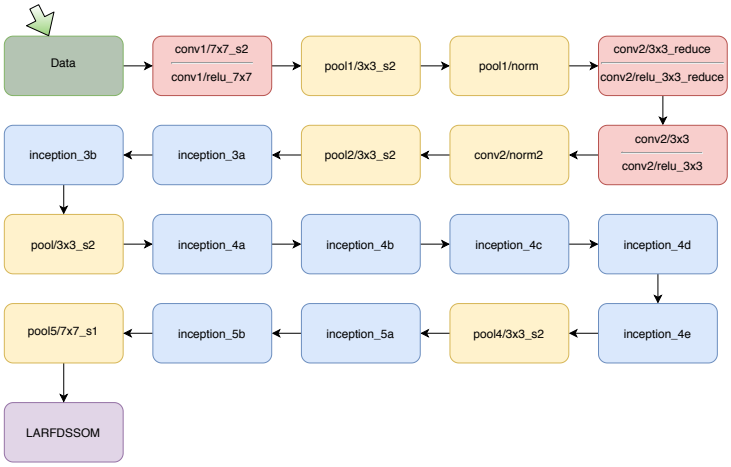


Figure 24: Proposed Architecture of the LARFDSSOM in MSSDC. We replaced the last classifier of the GoogLeNet Architecture and used the features extracted by the module in our LARFDSSOM clustering algorithm. In green is the input data, in red the convolutions, in yellow the pooling and normalization layers, in blue the inception module, and in purple LARFDSSOM.

4.2 DCSOM-RL IN MSSDC

In this scenario, we proposed a SOM layer implemented in PyTorch, so that it can be placed into DL models. The SOM layer developed is called SOM-RL, and it was designed to support batches of samples during its training as usually done in DL models. In our proposed model, SOM-RL was coupled in the latent space of the AE. The full framework with the AE is called DCSOM-RL, and it was designed to answer the following question: Is it possible to build a deep learning SOM layer that could be trained fully unsupervised? This step is important to remove the necessity of having the data labeled, a large amount of data available on the internet are not labeled, and making the clustering process driven by the data. Following, we explain how the layer was developed. The Figure 25 illustrates SOM-RL performing MSSDC.

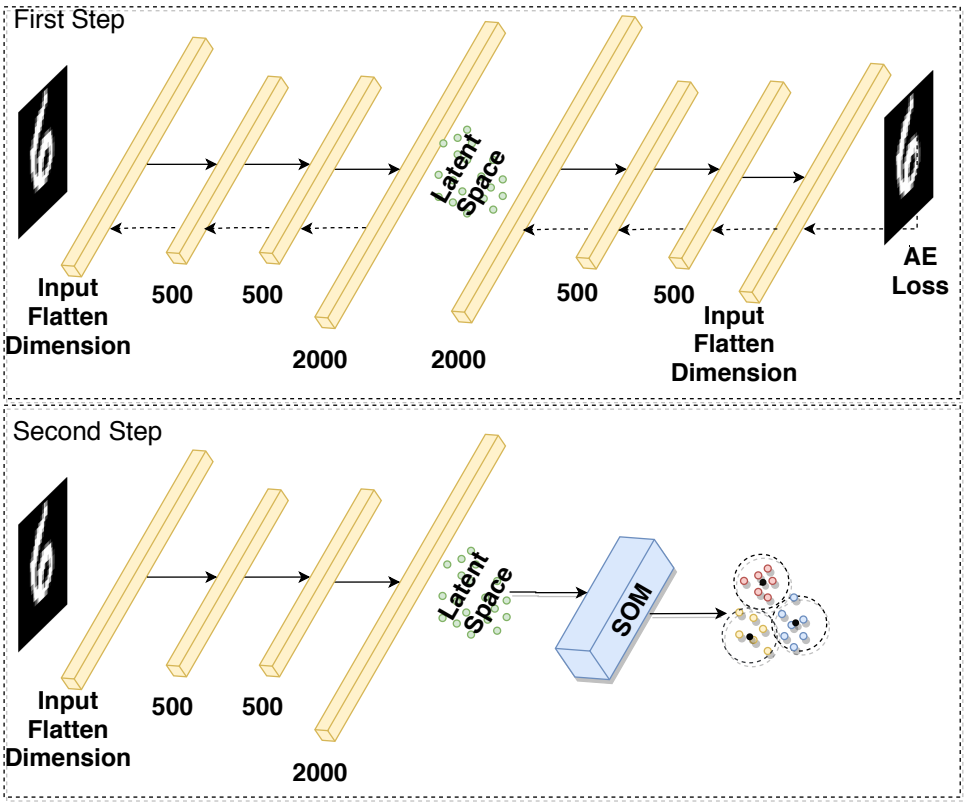


Figure 25: Proposed DCSOM-RL Model. In the first step, the Autoencoder is trained, and in the second step, the pre-trained encoder is used to guide the training of the SOM-RL model.

Differently from LARFDSSOM, to avoid having to keep track of node connections, SOM-RL presents a dynamic neighborhood that works as a radial basis function centered at the input pattern ($h(r) = exp(-\frac{r}{\gamma})$, where γ controls the decay rate and r is the activation rank of the node). It also has a node removal method that is more suitable for online learning, and implementation focused on parallelism that avoids sequential operations that are usually performed in SOM-based methods with loops and multiple conditions. Moreover, similarly to LARFDSSOM, the model also has a time-varying structure, a local receptive field that is adapted for each prototype that is correlated with its local variance and can learn different relevances for

each input dimension on each prototype. These characteristics are detailed in the next sections.

4.2.1 Nodes Structures of SOM-RL

Basically, as in LARFDSSOM (Bassani & Araujo, 2014), each prototype j in SOM-RL assumes the role of a cluster prototype, associated three m-dimensional vectors, where m is the number of input dimensions:

- **Center Vector:** $\mathbf{c}_j = \{c_{ji}, i = 1, \cdots, m\}$ represents the prototype of each cluster j.
- **Relevance Vector:** $\boldsymbol{\omega}_j = \{\omega_{ji}, i = 1, \cdots, m\}$, in which each component represents the estimated relevance, a weighting factor within $[0, 1]$, that the prototype j applies for the ith input dimension.
- **Distance Vector:** $\boldsymbol{\delta}_j = \{\delta_{ji}, i = 1, \cdots, m\}$ stores a moving average of the observed absolute distance between the input patterns and the center vector, i.e., $|\mathbf{x} - \mathbf{c}_j(n)|$. It is only used to update the relevance vector.

4.2.2 Competition in SOM-RL

In SOM-RL, each node in the map competes to become a winner and form clusters of input patterns. The input samples (assumed to be in $[\mathbf{0}, \mathbf{1}]$ interval) are processed in batches of arbitrary sizes. Whenever a batch is presented to the map, a competition begins by calculating the activation of each node j to each sample $\boldsymbol{x}$ in the batch (a parallel operation). The samples of the batch associated with each winner node are grouped, and the average position of the group is computed. This average is then used as an input pattern $\boldsymbol{x}$ to update the map. The activation is expressed by $ac(D_\omega(\boldsymbol{x}, \boldsymbol{c}_j), \boldsymbol{\omega}_j)$, and is computed as a radial basis function of a weighted distance $D_\omega(\boldsymbol{x}, \boldsymbol{c}_j)$ with the receptive field adjusted as a function of $\boldsymbol{\omega}_j$, as shown by (4.1).

$$ac(D_\omega(\boldsymbol{x}, \boldsymbol{c}_j), \boldsymbol{\omega}_j) = \frac{\sum_{i=1}^{m} \omega_{ji}}{\sum_{i=1}^{m} \omega_{ji} + D_\omega(\boldsymbol{x}, \boldsymbol{c}_j) + \varepsilon}, \tag{4.1}$$

where $D_\omega(\boldsymbol{x}, \boldsymbol{c}_j)$ is a distance function weighted by $\boldsymbol{\omega}_j$, calculated by the following Equation (4.2).

$$D_\omega(\boldsymbol{x}, \boldsymbol{c}_j) = \sum_{i=1}^{m} \omega_{ji}(x_i - w_{ji})^2. \tag{4.2}$$

Despite being the same equations used in other models, such as LARFDSSOM, it serves for a different purpose: the model aims to find not only the most activated node but also the intensity of each one. Such intensity will define the neighborhood to be updated as well as their intensities with (4.1).

4.2.3 Node Insertion and Update

In SOM-RL, after the competition, samples that the most active node presents an activation below a certain threshold parameter *at* defined by LHS sampling algorithm, the model creates a new node j with its center $\boldsymbol{c}_j$ initialized as $\boldsymbol{x}$, the relevance vector $\boldsymbol{\omega}_j$ initialized with ones, and the distance vector $\boldsymbol{\delta}_j$ with zeros. Winner nodes and neighbors with activation above *at* are updated towards $\boldsymbol{x}$, as per (4.3).

$$\boldsymbol{c}_j(n+1) = \boldsymbol{c}_j(n) + lr[(\boldsymbol{x} - \boldsymbol{c}_j(n))], \tag{4.3}$$

where lr is the learning rate, which is maximum for the winner and smaller for the neighbors, $\boldsymbol{c}$ is the center of the node in the map.

In order to update the relevance vector $\boldsymbol{\delta}_j$, first, the average distance of each node to the input patterns it clusters is estimated. This is done by computing a moving average of the observed distance between the input pattern and the current center vector:

$$\boldsymbol{\delta}_j(n+1) = (1 - \beta * lr)\boldsymbol{\delta}_j(n) + (\beta * lr)|\boldsymbol{x} - \boldsymbol{c}_j(n+1)|, \tag{4.4}$$

where lr is the learning rate used in (4.3), $\beta \in]0,1]$ controls the rate of change of the moving average, and the operator $|\cdot|$ denotes the absolute value applied to the components of the vector.

Each component ω_{ji} of the relevance vector is calculated by an inverse logistic function of the distances δ_{ji}, as per (4.5).

$$\omega_{ji} = \begin{cases} \dfrac{1}{1 + \exp\left(\frac{\delta_{ji\text{mean}} - \delta_{ji}}{s(\delta_{ji\text{max}} - \delta_{ji\text{min}})}\right)} & \text{if } \delta_{ji\text{min}} \neq \delta_{ji\text{max}} \\ 1 & \text{otherwise,} \end{cases} \tag{4.5}$$

where $s > 0$ controls the slope of the logistic function. The relevances go to zero for dimensions with distances close to the maximum $\delta_{ji\text{max}}$, whereas in the other dimensions, they are set within [0, 1]. At the end of the training, we compute the activations among the remaining nodes to define connection and relations between them, and, thus, the final topology of the map. The nodes that are mutually activated, above a_t threshold, are connected.

4.2.4 Node Removal in SOM-RL

In SOM-RL, we introduce a concept of life to each node. They begin at 100% and lose their vitality by a parameter factor $ld \in [0\%, 100\%]$ every time they do not win a competition, i.e., present an activation that is not the highest. Therefore, whenever a node achieves a life value of 0, it is removed from the map. Alternatively, the life of a node is restored to 100% when it wins a competition.

4.2.5 Neighborhood in SOM-RL

The neighborhood of SOM-RL is not fixed but defined dynamically at each update step as a function of the nodes activation rank, from the highest to the lowest. Therefore the neighbors of the winner node are also updated towards the input pattern, but with lower learning rates since it decays exponentially as a function of the position of the neighbor in the rank, as $h(r) = exp(-\frac{r}{\gamma})$, where r is the rank, and γ a parameter that controls the decay rate.

Before updating the nodes, the final learning rate is computed by $lr * h(r)$. Notice that the most active node will be fully updated according to lr since $h(r)$ is 1 for rank 0 and decays as the rank increases and activation decreases.

Therefore, as the neighborhood of SOM-RL is not related to the nodes themselves but to the input patterns presented to the model, a sample-driven approach.

4.2.6 SOM-RL Forward Pass Algorithm

The SOM-RL training procedure is described in Algorithm 1. Note that the variable n is used to control the current number of nodes in the map, and n_{max} defines its limit. It is used solely for implementation purposes, once we have to define a fixed shape for the matrices and vectors to avoid concatenation operations, that leads to a loss of performance. The operations to control the active nodes are performed by using logical masks.

At this point, the SOM-RL can perform clustering following the next steps: First, we train an autoencoder to create a representation in the latent space (unsupervised learning). Second, we extract the features with the encoder. Third, we train the SOM-RL with these features.

4.3 DCSOM-RL IN JDC

In this scenario, we use the SOM-RL layer to compute the unsupervised error (error between the prototype of SOM and the input data) and backpropagate it to adjust the AE weights, through a loss that combines both AE and SOM errors. The DCSOM-RL, illustrated in Figure 26, can propagate the unsupervised error to adjust artificial neural network weights to train both models simultaneously in an end-to-end unsupervised fashion. This will allow us to learn mappings from a complex to a simple disentangled representation space that SOM can handle with smooth transitions between cluster points. With this approach, we expect that the model can learn a better representation with smooth transitions between the cluster regions.

Algorithm 1: SOM-RL Forward Pass

Input : Batch of Input Patterns $\boldsymbol{x}$

Require : Initialize parameters a_t, lr, ρ, ld, β, s, n_{max}

Function `SOM-RL` ($\boldsymbol{x}$) **:**

- **if** *map is empty* **then**
  - Initialize the map with one node with $\boldsymbol{c}_j$ initialized at $\bar{x}$, $\boldsymbol{\omega}_j \leftarrow \mathbf{1}$, $\boldsymbol{\delta}_j \leftarrow \mathbf{0}$, $\text{life}_j \leftarrow 1$;
  - Set the number of nodes $n \leftarrow 1$;
- **else**
  - Compute the activations $\mathbf{A}$ of all nodes for each $\boldsymbol{x}_i \in \boldsymbol{x}$ (4.1);
  - Find the winners s_i with the highest activation for each $\boldsymbol{x}_i \in \boldsymbol{x}$;
  - **forall** $s_i \in \boldsymbol{s} \mid A_{i,s_i} < a_t$ ***and*** $n < n_{max}$ **do**
    - Create a new node j and set: $\boldsymbol{c}_j \leftarrow \bar{x}_{s_i}$, $\boldsymbol{\omega}_j \leftarrow \mathbf{1}$, $\boldsymbol{\delta}_j \leftarrow \mathbf{0}$, $\text{life}_j \leftarrow 1$;
    - Set $n \leftarrow n+1$;
  - **forall** $s_i \in \boldsymbol{s} \mid A_{i,s_i} \geq a_t$ **do**
    - Compute the learning rates $\boldsymbol{lr}_n$ of the neighbors by ranking $\mathbf{A}_i$;
    - Update the winner nodes and its neighbors towards $\bar{x}_{s_i}$ with lr and $\boldsymbol{lr_n}$, respectively:
    - Update the distance vectors $\boldsymbol{\delta}_{s_i}$ and $\boldsymbol{\delta}_n$ (4.4);
    - Update the relevance vectors $\boldsymbol{\omega}_{s_i}$ and $\boldsymbol{\omega}_n$ (4.5);
    - Update the center vectors $\boldsymbol{c}_{s_i}$ and $\boldsymbol{c}_n$ (4.3);
    - Decrement the life of all nodes $j \notin \boldsymbol{s}$ to $\text{life}_j \leftarrow \text{life}_j - ld$;
  - Remove all nodes j with $\text{life}_j < 0$; Decrement n according to number of removed nodes;

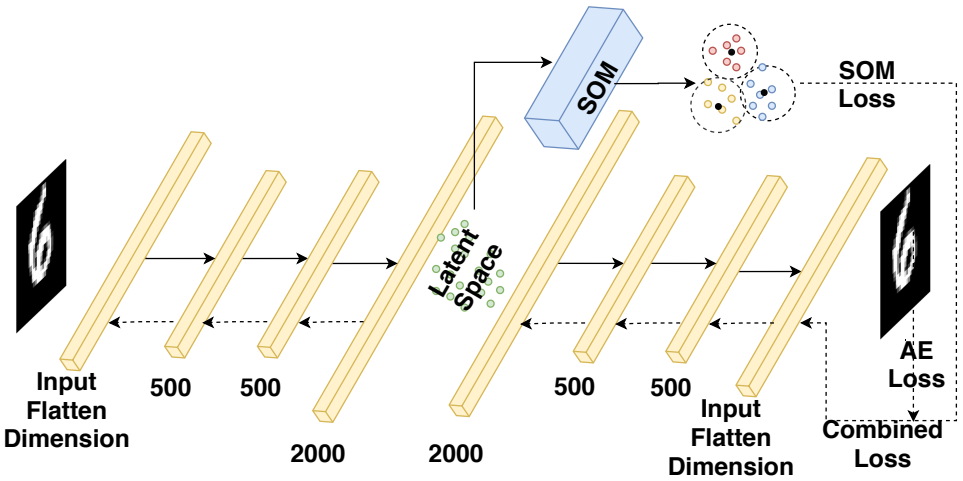


Figure 26: Proposed DCSOM-RL Model. In the first step, the Autoencoder and the SOM-RL are trained jointly.

Here, the AE is architecture used is based on Xie *et al.* (2016). To achieve this, we designed a new loss function based on the distance between the input sample and the winner prototype that can be used to backpropagate errors to previous layers.

Lets denote the encoder and decoder parameters as θ_e and θ_d, respectively. An input $\mathbf{x} \in \mathbb{R}^d$ is mapped to a latent space encoding $\mathbf{z}_e \in \mathbb{R}^m$. Bassani & Araujo (2014) have shown

that it is easier to adjust the parameters of SOM when the input dimensions are scaled to the [0, 1] interval. To maintain this behavior, $\boldsymbol{z}_e$ is computed by $\boldsymbol{z}_e = sigmoid(f_{\theta_e}(\boldsymbol{x}))$. The encoding is then fed as input to a forward pass of SOM-RL and a decoder pass of the AE architecture. At this point, a reconstruction $\hat{\boldsymbol{x}}$ of the input can be computed as $\hat{\boldsymbol{x}} = g_{\theta_d}(\boldsymbol{z}_e)$. This component is used to calculate the reconstruction loss $\mathscr{L}_{\text{rec}}(\boldsymbol{x}, \hat{\boldsymbol{x}}) = \|\boldsymbol{x} - \hat{\boldsymbol{x}}\|^2$. It is well-know that AE can minimize the reconstruction error ensuring that hidden units capture the most relevant aspects of the data (Murphy, 2012).

In SOM-RL, a competition starts to find the winner as the most active node for the latent representation in $\boldsymbol{z}_e$, when it is fed as input, as per Equation (4.1). Then, the winner and its neighbors move towards the input by updating the center vectors, and the new relevance vectors are estimated. Notice that when there is no node sufficiently activated, according to a threshold parameter, SOM-RL will create and insert new nodes into the map due to its time-varying structure, at the position of the input pattern. At the end of this process, the winner prototype for $\boldsymbol{z}_e$, $\boldsymbol{c}_z$ is used to compute the clustering loss, defined as $\mathscr{L}_{SOM-RL}(\boldsymbol{z}_e) = D_{\omega}(\boldsymbol{x}, \boldsymbol{c}_z)$, that is the distance weighted by the each dimension relevances found by SOM-RL calculated with Equation (4.2). It is expected that $\mathscr{L}_{SOM-RL}$ creates a tendency to approximate encodings to prototypes. Due to the latent space constraints (all dimensions between [0, 1]), and the fast convergence characteristics of SOM-RL, even with a small number of samples, $\mathscr{L}_{SOM-RL}$ will frequently converge to small values in few epochs.

To achieve the main challenge of optimizing our complete framework, but more precisely the representations (prototypes) learned by our SOM-RL, the AE is trained jointly with the SOM-RL. To do so, we add the $\mathscr{L}_{SOM-RL}$, weighted by an α, to the $\mathscr{L}_{\text{rec}}$ in order to add the gradient information about $\boldsymbol{z}_e$ with respect to the SOM-RL state. The complete loss function to be minimized is given by Equation (4.6):

$$\mathscr{L}_{DCSOM-RL}(\boldsymbol{x}, \hat{\boldsymbol{x}}) = \mathscr{L}_{\text{rec}}(\boldsymbol{x}, \hat{\boldsymbol{x}}) + \alpha \mathscr{L}_{SOM-RL}(\boldsymbol{x}_z). \tag{4.6}$$

This chapter explained the proposed models in each scenario: (i) LARFDSSOM in MSSDC, (ii) SOM-RL in MSSDC and (iii) SOM-RL in JDC. Next, we are going to validate the models with several experiments to consolidate each model and build the necessary understanding of each designed scenario to accomplish our main objective to train a SOM layer in a fully unsupervised way with images.

5 EXPERIMENTS AND RESULTS

This chapter describes experiments that aim to validate the proposed models, and it is divided as follow: First we present the metrics (Section 5.1), the dataset benchmarks (Section 5.2) and the parameters setting (Section 5.3), subsequently, we validate the work in different scenarios (LARFDSSOM in MSSDC (Section 5.4), DCSOM-RL in MSSDC (Section 5.5) and DCSOM-RL in JDC (Section 5.6)).

5.1 METRICS

In this section, we present the metrics used in this work. First, we consider the *clustering error* (CE)(Section 5.1.1) metric for the LARFDSSOM scenario, because it is currently used in recent literature of SOM models such as Bassani & Araújo (2012); Bassani & Araujo (2014); Braga & Bassani (2018, 2019). Purity (Section 5.1.2) and NMI (Section 5.1.3) datasets are commonly used in DC, so we bring them to validade our DC model and compare it to SOM with DC benchmarks (Fortuin *et al.*, 2018; Forest *et al.*, 2019). It is important to mention that the analisis of the NMI is essential because we can measure the amount of information is being described by the clusters.

5.1.1 Clustering Error (CE)

Among the metrics described and analyzed in Patrikainen & Meila (2006) and Müller *et al.* (2009), the CE has been used to evaluate SOM-based models. The following definition proposed by Patrikainen & Meila (2006) is important to understand the CE metric: A clustering C is a partitioning of the set of m data points into disjoint clusters $C_1, C_2, ..., C_K$ of sizes $m_1, m_2, ..., m_K$, where $\sum_i m_i = m$. Assume that we have two clusterings $C = \{C_1, C_2, ..., C_K\}$ and $C' = \{C'_1, C'_2, ..., C'_{K'}\}$. The confusion matrix $M = (m_{ij})$ is a $K \times K'$ matrix whose $ij-th$ element is the number of points in the intersection of clusters C_i and C'_j, i.e., $m_{ij} = \left|C_i \cap C'_j\right|$.

For computing CE, consider a clustering algorithm that has found C' clusters, when the correct number of categories in the dataset is C. The clustering is represented by a confusion matrix $M_{C' \times C}$. The clusters are matched to the ground-truth labels by finding the row or column permutation of M that maximizes the sum of its diagonal elements, solved in polynomial time by the Hungarian method (Kuhn, 1955), after ensuring that the confusion matrix is a square matrix by inserting rows or columns filled with zeros. After that, the final CE measure is given by the following equation:

The name of the metric (CE) is unusual in the ML community because the idea of error is ideal with something that we want to minimize. Considering this, we are going to adopt in this work the CE as a metric that is going to be minimized differing from the original metric.

$$CE(C,C') = \frac{|U| - D_{max}}{|U|}, \tag{5.1}$$

where $D_{\max}$ is the maximized sum of the diagonal elements of M and $|U|$ is the number of data matrix elements in the union of C and C'. CE is a metric that takes into account not only the produced groupings but also the relevant dimensions found for each group and penalizes results with too many groupings. CE is calculated as a percentage of points that are grouped differently, considering an optimal coincidence between the desired groupings and those obtained. The metric used ranges from zero to one and higher values indicate better groupings, despite its misleading name.

5.1.2 Purity

A clustering result satisfies homogeneity if all of its clusters contain only data points that are members of a single class. Purity is a homogeneity metric of a cluster labeling given ground-truth labels. The metric is the maximum class probabilities for the ground-truth category labels X and obtained cluster labels Y. Given variables (x,y) sampled from the finite discrete joint space $X \times Y$ (Tuytelaars *et al.*, 2010). The metric is given by the following Equation (5.2):

$$Purity(X|Y) = \sum_{y \in Y} p(y) \max_{x \in X} p(x|y). \tag{5.2}$$

5.1.3 Normalized Mutual Information (NMI)

The Mutual Information is a symmetric measure $I(X|Y) = I(Y|X)$ to quantify the statistical information shared between two distributions (X and Y) (Cover & Thomas, 1991). The following Equation (5.3), describes the mutual information of two random variables, where $H(.)$ is the entropy:

$$I(X|Y) = H(X) - H(X|Y). \tag{5.3}$$

The *normalized mutual information* (NMI) (Strehl & Ghosh, 2002) provides an indication of the shared information between a pair of clusters. The metric varies between 0 (no mutual information) and 1 (perfect correlation) due to its normalization. Equation (5.4) describes the arithmetic normalization of the NMI metric:

$$NMI(X|Y) = \frac{I(X|Y)}{\frac{1}{2}\left[H(Y) + H(X)\right]}, \tag{5.4}$$

where Y denotes the ground-truth labels, X denotes the clusters labels, $I(X|Y)$ is the mutual information metric between X and Y, and $H(.)$ is the entropy. Equation (5.5) describes the geometric normalization version:

$$NMI(X|Y) = \frac{I(X|Y)}{\sqrt{H(X)H(Y)}}. \tag{5.5}$$

In this work, we choose the NMI arithmetic version as the NMI metric due to the fact that the scikit-learn implementation of the metric is more stable than the geometric.

NMI is a more balanced measure for clustering performance than purity due to the penalty term for the number of clusters. Purity may lead to a scenario in which results that present a high number of clusters are rewarded in detriment of more meaningful representations.

5.2 BENCHMARK DATASETS

In this section, we present the benchmark datasets used in this work. The Caltech-256 (Section 5.2.1) is used in the UVOC field as an alternative to ImageNet. We used this dataset to evaluate clustering algorithms such as k-means, SOM, LARFDSSOM. MNIST (Section 5.2.2) and Fashion-MNIST (Section 5.2.3) datasets are commonly used in DL, so we bring them to validade our DC model.

5.2.1 Caltech-256

The Caltech-256 dataset has 256 categories with 30,507 colored images with different sizes, Figure 27 illustrates some samples of this dataset. In this work, we used the Easy Subset of Caltech-256 comprising 20 hand-picked categories by Tuytelaars *et al.* (2010) (Table 1), twelve other 20-category subsets from Caltech-256 (first subset was taken from categories 1 to 20, second from 21 to 40 and so on)[1].

Table 1: Easy Subset - Categories selected by Tuytelaars *et al.* (2010).

American flag	diamond ring	dice	fern
fireworks	French horn	ketch 101	killer whale
mandolin	motorbikes 101	pci card	rotary phone
tombstone	Pisa tower	zebra	airplanes 101
fire extinguisher	leopards 101	roulette wheel	faces easy 101

5.2.2 MNIST

The MNIST dataset is a handwritten digits dataset commonly used for evaluating machine learning algorithms. It is composed of a training set of 60,000 samples and a test set of 10,000 samples. Introduced by LeCun (1998), the dataset is used nowadays as a standard dataset to evaluate novel deep learning techniques due to its simplicity and wide usage by the research

[1] All subset lists can be found in `http://homes.esat.kuleuven.be/~tuytelaa/unsup_data.html`

(a) American flag

(b) Diamond ring

(c) Dice

Figure 27: Examples of Easy Subset of Caltech-256: The images have different sizes and represent only one class per image.

community. This dataset was built as a subset of the NIST images pre-processed as the following method to become the MNIST dataset: the images were centered in a 28×28 image by computing the center of mass of the pixels. After that, the images were translated to position this point at the center of the 28×28 field (LeCun *et al.*, 1998). Figure 28 illustrate images from MNIST dataset.

Figure 28: Examples from the MNIST dataset.

5.2.3 Fashion-MNIST

The Fashion-MNIST dataset is based on fashion products from Zalando's research. It has different photos of many products, for example, trousers, coats, and bags. Xiao *et al.* (2017) selected the images and pre-processed to make them similar to the format of MNIST. The Fashion-MNIST has $70,000$ 28×28 grayscale images of 10 categories of fashion products, with $7,000$ images per category. The training set has $60,000$ images, and the test set has $10,000$ images. Figure 29 illustrates images from the MNIST dataset.

Figure 29: Examples from the Fashion-MNIST dataset.

5.3 PARAMETERS TUNING

The parameter tuning was used to explore the parameter's sensibility and how each parameter impacts the proposed models. When the parameters are tuning for different scenarios, we can evaluate the range that the parameter works for that task and if it positively or negatively influences the model. We can also use parameter tuning to remove parameters that do not impact the model. In our work, we applied a statistical method for parameter sampling for the parameters tuning of our experiments. The method used is the *latin hypercube sampling* (LHS) (McKay *et al.*, 2000; Helton *et al.*, 2005), it guarantees the full coverage of the range of each parameter in the parameter space.

Let $\mathbf{x} = \{X_1, ..., X_k\}$ be the values of k input parameters, and S be the sample space of $\mathbf{x}$. The LHS aims to represent all areas of the sample space S of $\mathbf{x}$ partitioning S into $\mathbf{L}$ disjoint intervals. It ensures that all portions of S are sampled and that each of the input parameters, X_i has all portions if its distributions represented by input values. To make it attainable, it splits the range of each parameter X_i into N intervals of the same probability of $1/N$, resulting in a random selection of a single value from each interval. After that, each sampled component from X_i is matched at random with the other various X_i. LHS ensures that each component is represented, no matter the importance that a component might have (McKay *et al.*, 2000; Helton *et al.*, 2005).

5.3.1 SOM-RL Parameters

Batch Size, **Epochs**, and **SOM learning rate** (lr), by intuition, do not need further explanation. For fair comparisons with state of the art competitors, we fixed the batch size to 256. Moreover, the number of epochs was defined at first hand to range from 10 to 1,000. Nonetheless, an extensive empirical analysis showed that 30 epochs are more than enough for the model to stabilizes.

Clustering Loss weighting factor (α). It is the weighting factor that penalizes the

clustering loss. It is an empirical value that is used to control the influence (or contribution) of the clustering loss to the final loss.

SOM-RL input dimensions. It is the number of dimensions given as input to the model. It defines the dimensionality of the latent space after the input is encoded.

Activation threshold (a_t). Activation threshold. During training, if the activation of the winner node is below this level, a new node is inserted to define a new cluster.

Life decay (ld). It is the life decreasing rate that is applied when a node does not win a competition. Whenever the life of a node reaches zero, the node is removed from the map.

Relevance rate (β). Rate of change of the moving average used to compute the relevance vector. Higher values make the nodes to adapt faster to the relevant dimensions. Too high values provoke instability. Lower values produce a smoother adaptation.

Neighborhood decay (γ). This parameter controls how strong will be the update of neighbors towards the input pattern. The winners are fully updated according to the lr, but the neighbors are updated with a lower factor, according to their activation. Distant nodes have a low activation and are barely influenced.

Relevance smoothness (s). It represents the slope of the logistic function when calculating relevances. Values close to zero, produce a sharp slope. As the value increases, the slope becomes less prominent and all relevances tend to be similar. Values higher than 1.0 result in similar relevances values around 0.5 for all dimensions.

Max number of nodes (n_{max}). This value should be higher than the number of expected clusters in the dataset to allow exploration of the search space. It also may be used to prevents trivial solutions, suchlike one cluster for each data point, and to control memory usage. It is important to mention that this parameter, when other parameters are set correctly it should not influence significantly the outcome due to the time-varying topology of the model that inserts and removes nodes when it is necessary during the self-organizing process.

5.3.2 Parameter Ranges

In the first scenario, while AP and k-means have only one parameter to be tuned, SOM, LDA and LARFDSSOM have 5, 7 and 8 respectively. Thus we followed the LHS to test several different parameter combinations. First, we define a wide sampling range for all parameters and run each method with 100 LHS samples drawn inside the range. Then we repeat this process narrowing the sampling ranges to the observed sweet spots of each parameter. For k-means, the number of the clusters in the dataset, k, is the only parameter. Therefore, we run k-means with values for k varying around the actual number of clusters in the dataset (i.e., ranging from 10 to 30 in a 20 categories datasets). With k-means, we achieve 100 samples by running the method 5 times with each of the 20 values chosen for k, with a different random seed. Table 2 shows the range of parameters used for the LARFDSSOM.

In the second scenario we run over the LHS for a wider range with 50 LHS points. For

Table 2: Parameter Ranges for LARFDSSOM. Source: Bassani & Araujo (2014).

Parameters	**min**	**max**
Activation threshold (a_t)	0.7	0.999
Lowest cluster percentage (lp)	0.1%	10%
Relevance rate (β)	0.001	0.1
Max competitions $(maxcomp)$	$1 \times S^*$	$100 \times S^*$
Winner learning rate (e_b)	0.001	0.1
Neighbors learning rate (e_n)	$0.0001 \times e_b$	$0.5 \times e_b$
Relevance smoothness (s)	0.01	0.1
Connection threshold (c)	0	0.5

*S is the number of input samples in the dataset.

the adjusted ranges, an adjusted range from the tendency analysis of the metrics and parameters, we run for 30 different sets of parameters. Finally, for this scenario, we run only one experiment with the adjusted set up. Table 3 shows the range of parameters used for SOM-RL in this scenario.

Table 3: Parameter values starting from initial ranges, then an intermediate step of Adjusted Ranges, and finally, the best values for each dataset. The best-reported values come from the adjusted ranges.

Parameters	Initial Ranges		Adjusted Ranges		**Best Values**
	min	**max**	**min**	**max**	**MNIST**
Batch Size	16	512	256	-	256
SOM input dimensions	10	100	10	-	10
Activation threshold (a_t)	0.85	0.999	0.95	0.999	0.979
SOM learning rate (lr)	0.0001	0.01	0.0001	0.003	0.001
Life decay (ld)	0.05	0.5	0.1	0.5	0.2255
Relevance rate (β)	0.001	0.5	0.005	0.5	0.15845
Neighborhood decay (γ)	0.1	4.0	0.1	4.0	0.8734
Relevance smoothness (s)	0.01	0.1	0.05	0.1	0.0505
Max number of nodes (n_{max})	10	150	10	80	64

In the third scenario, Table 4 shows the range of parameters used for SOM-RL with JDC. For initial ranges, the model was executed 30 times 30 different parameter settings. For adjusted ranges, the model was run 10 times with 10 different sets of parameters. Finally, for the final round of experiments, we fixed the best values varying the seed over 10 runs to extract the mean and standard deviation.

Table 4: Parameter values starting from initial ranges, then an intermediate step of Adjusted Ranges, and finally, the best values for each dataset. The best-reported values come from the adjusted ranges.

Parameters	Initial Ranges		Adjusted Ranges		**Best Values**	
	min	**max**	**min**	**max**	**MNIST**	**Fashion-MNIST**
Batch Size	16	512	256	-	256	256
Epochs	10	1,000	30	-	30	30
$\mathscr{L}_c$ weighting factor (α)	0.01	1.00	0.01	0.15	0.1145	0.024
SOM input dimensions	10	50	20	30	22	22
Activation threshold (a_t)	0.95	0.999	0.99	0.999	0.99425	0.99885
SOM learning rate (lr)	0.001	0.3	0.05	0.15	0.1426	0.1383
Life decay (ld)	0.01	0.6	0.1	0.5	0.36	0.46
Relevance rate (β)	0.01	0.5	01	0.15	0.067	0.0195
Neighborhood decay (γ)	0.17	7.0	2.0	6.0	1.382	4.916
Relevance smoothness (s)	0.01	0.7	0.4	0.7	0.426	0.658
Max number of nodes (n_{max})	20	300	50	250	200	200

5.4 SOM AND LARFDSSOM IN MSSDC

In this section, we present the evaluation of the clustering algorithms in the MSSDC scenario. It is important to remember that this task worked with transfer-learning, which we extracted the Caltech-256 features using GoogLeNet pre-trained in ImageNet. Due to this, the input images had to be resized for 224×224. We divided the section into two parts: the first one to present the results of the model and the second one to analyze and discuss it.

We compare LARFDSSOM with other clustering algorithms applied for UVOC in MSSDC: k-means, Latent Dirichlet Allocation (LDA) (Blei *et al.*, 2003), SOM (Kohonen, 1990) and Aggregated Partition (AP) (López-Sastre, 2016). The k-means and LDA implementations were obtained from the scikit-learn Python package and LARFDSSOM from Bassani & Araujo (2014) C++ implementation. For AP, we ran the Matlab code provided by the authors. The results reported are the mean CE of 10 runs with the best parameter sample found with LHS.

First, we evaluated the influence of image representation on the clustering performance. We assessed SIFT features on interest points detected by Harris-Laplace, Hessian-Laplace, Hessian-Affine, and dense detectors, along with deep CNN features obtained with GoogLeNet (Szegedy *et al.*, 2015). The SIFT features were extracted using the implementation provided by collaborative work between the Visual Geometry Group, Katholieke Universiteit Leuven, Inria Rhone-Alpes, and the Center for Machine Perception [1].

[1]SIFT features implementation: `http://www.robots.ox.ac.uk/~vgg/research/affine/index.html`

5.4.1 Results

In Table 5, we used the Caltech-256 Easy Subset to compare the different clustering algorithms over different image descriptions with the CE metric. Next, in Table 6, we evaluated all subsets proposed by Tuytelaars *et al.* (2010) with CE, extending our previous result using more subsets of Caltech-256, and analysing the most relevant parameters of the LARFDSSOM: The activation parameter (a_t), the winner prototype learning rate (e_b) and the neighborhood prototypes learning rate (e_n).

Table 5: Clustering error followed by standard deviation of different algorithms over the Easy subset of Caltech-256. Images are represented with SIFT descriptor (combined with several interest point detectors) or GoogLeNet. Lower is better. In bold, the best result for each descriptor. Algorithms marked with an asterisk are statistically equal to the best performing one. In parentheses, the number of categories found. Combo features refer to the combination of HarLap+Heslap+dense features. These CE results were evaluated by us.

	LARFDSSOM	AP	SOM	k-means	LDA
HarLap	**0.446 ± 0.000** (17)	0.725 ± 0.008 (147)	0.542 ± 0.000* (26)	0.579 ± 0.009* (15)	0.674 ± 0.011 (19)
HesLap	0.578 ± 0.000 (6)	0.767 ± 0.004 (136)	**0.576 ± 0.000** (16)	0.597 ± 0.000 (17)	0.725 ± 0.011 (21)
Dense	**0.629 ± 0.010** (17)	0.752 ± 0.001 (101)	0.583 ± 0.000* (14)	0.634 ± 0.014* (13)	0.634 ± 0.003* (19)
Combo	0.530 ± 0.000* (11)	0.730 ± 0.002 (90)	**0.526 ± 0.000** (27)	0.583 ± 0.025* (10)	0.707 ± 0.035 (17)
HesAff	**0.596 ± 0.020** (13)	0.756 ± 0.003 (103)	0.616 ± 0.025* (13)	0.643 ± 0.023 (10)	0.706 ± 0.016 (17)
GoogLeNet	**0.044 ± 0.000** (27)	0.179 ± 0.005 (35)	0.133 ± 0.000 (21)	0.185 ± 0.067 (19)	0.130 ± 0.042* (23)

Table 6: Clustering error followed by standard deviation and the best parameters of LARFDSSOM. Images are represented with GoogLeNet features. Lower is better. The correct number of categories for each set is in brackets on the first column. In CE column, the number in brackets represents the number of clusters found. These CE results were evaluated by us.

Dataset	**CE**	$\mathbf{a_t}$	$\mathbf{e_b}$	$\mathbf{e_n}$
Easy (20)	0.037 ± 0.000 (20)	0.9897849	0.043	0.222
Subset1 (20)	0.155 ± 0.008 (22)	0.972673	0.096	0.031
Subset2 (20)	0.101 ± 0.007 (21)	0.986542	0.058	0.020
Subset3 (20)	0.129 ± 0.010 (24)	0.968689	0.053	0.039
Subset4 (20)	0.087 ± 0.013 (22)	0.983551	0.149	0.075
Subset5 (20)	0.094 ± 0.007 (22)	0.982757	0.049	0.039
Subset6 (20)	0.132 ± 0.009 (19)	0.987612	0.073	0.095
Subset7 (20)	0.143 ± 0.019 (20)	0.991315	0.091	0.045
Subset8 (20)	0.099 ± 0.016 (20)	0.980394	0.055	0.054
Subset9 (20)	0.130 ± 0.012 (20)	0.972480	0.111	0.030
Subset10 (20)	0.149 ± 0.007 (19)	0.977887	0.075	0.056
Subset11 (20)	0.114 ± 0.021 (21)	0.982614	0.084	0.023
Subset12 (20)	0.158 ± 0.016 (23)	0.973440	0.112	0.041

5.4.2 Analysis of the Results

We focused on UVOC, covering image description methods. For the first time, LARFDS-SOM was evaluated in this challenging field, obtaining the best CE results in clustering. We performed a brief image description evaluation, comparing SIFT descriptors combined with several standard interest point detectors, and GoogLeNet, a DNN approach to image categorization. The results showed that the GoogLeNet-based image description allowed for significantly better clustering results than its SIFT counterpart.

Table 6 shows that the optimum value of e_n tends to increase slightly for easier datasets, while the reverse also holds (see Easy rows). It might be because, in easy datasets, class clusters are much more compact (i.e., images that belong to the same class are close to each other in the feature space), thus higher e_n may induce this kind of clustering behavior. However, other parameters do not show such a straightforward relation regarding dataset difficulty.

Overall, the results obtained in UVOC were consistent and precise (as low as 10% of error in certain subsets). This means that we achieved data assignments considerably similar to those made by humans, a relevant achievement for an unsupervised learning method. We attribute this primarily to the subspace clustering capabilities of LARFDSSOM (i.e., relevance learning), because this is the most notable difference between SOM and LARFDSSOM when the number of nodes of SOM is set to the number of clusters, and yet, LARFDSSOM has been significantly better than SOM in such conditions. These results show the powerful capability of using LARFDSSOM in combination with DL methods of representation learning. This is a good indication that using a SOM-layer in the DC scenario can be a promising approach. This is evaluated in the next sections.

5.5 DCSOM-RL IN MSSDC

In this scenario, we evaluate the developed SOM layer with an AE, which in AE replace the classifier model. The following experiments evaluate the results from quantitative and qualitative perspectives of our SOM layer implemented in the PyTorch framework. For the first, two common metrics were used on the MNIST dataset: NMI and Purity, following the same experimental setup of DE-SOM. For the latter, the quality of the latent representations was explored using the relations between the encoded features of the dataset and the prototypes of DCSOM-RL, and the top ten samples closest to each DCSOM-RL prototype.

The setup was based on Forest *et al.* (2019) to permit one-to-one comparisons. The models were trained for $1,000$ iterations with a batch size of 256, as in DE-SOM. The AE is consistent with the architecture proposed by Xie *et al.* (2016). A latent space of 10 features is obtained after passing through the encoder stage, which finishes with a sigmoid function. The encoded sample is then fed as input to SOM-RL. The maximum number of prototypes is 64 to be consistent with the 8×8 grid defined in Forest *et al.* (2019). The α parameter varies between

0 and 1, but the best value for this scenario was 0.001.

5.5.1 Quantitative Analysis

Table 7: Evaluation metrics on MNIST dataset: Purity and NMI for the best parameters of DCSOM-RL compared with k-means and DC SOM benchmarks. Higher is better. Only the DCSOM-RL was evaluated by us.

METHOD	MNIST	
	PURITY	NMI
K-MEANS	0.791	0.537
SOM-VAE	0.868	0.595
DE-SOM	0.939	0.657
SOM-RL	0.921	0.615

Table 7 shows the clustering quality in terms of NMI and purity of DCSOM-RL in comparison with DE-SOM and SOM-VAE. The results show that the proposed model achieves competitive results while reducing the dimensionality from 784 input features to 10 features in latent space. However, the results are close, showing a promising path to follow. It is important mentioning that the main idea is not necessarily to outperform in terms of metric value but to build solid representations with meaningful topological properties that can not be achieved with a dimensional grid.

5.5.2 Qualitative Analysis

We analyzed the reconstructed image of each prototype and its top ten closest samples. In Figure 30, notice that the prototypes shown represent two ways of writing the numbers of MNIST dataset, except number one that only had one prototype representing it. In that figure, we see some differences that the prototypes catches like in number 6 prototypes with angle variation when analyzing the two different prototypes' angle. We can also see the different prototypes representing the number 2, got different writing ways of round the bottom part of this number.

Then, we evaluate a plot in t-SNE space (Maaten & Hinton, 2008) of the datapoints in relation to the cluster prototypes for the test samples (Figure 31). The t-SNE is a nonlinear dimensionality reduction technique for visualization of high-dimensional data in a low-dimension space, and similar objects are modeled closer in this space. t-SNE was chosen because our map does not present a 2D disposal grid, as the related competitors. The edges between prototypes represent topological neighborhood found, i.e., connected prototypes are considered somehow similar by the model. Notice that the model was able to create at least one cluster for each class with some connections among prototypes representing the same class and it rarely has connections with prototypes of different classes. The observed connections between nodes of

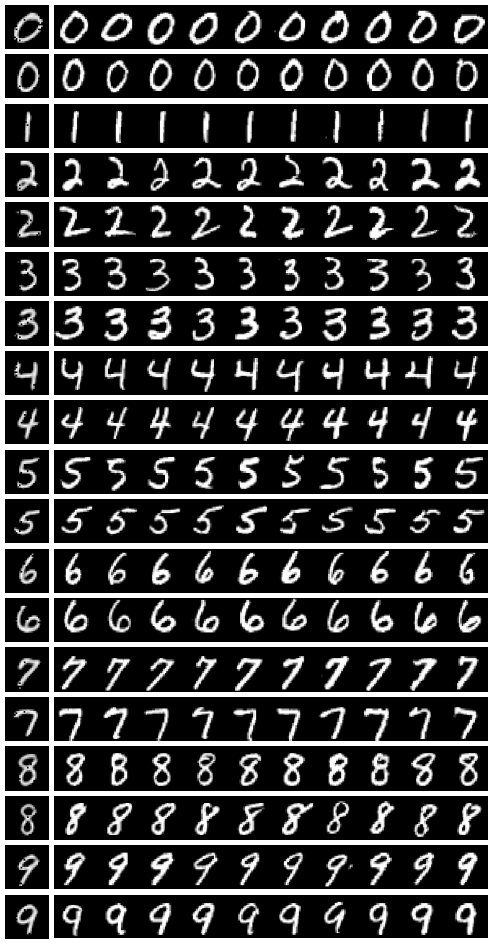


Figure 30: On the left, a column with the prototypes. On the right, the top ten samples closest to the prototypes. We selected two prototypes for each MNIST dataset class, except for the number one that only had one prototype representing it.

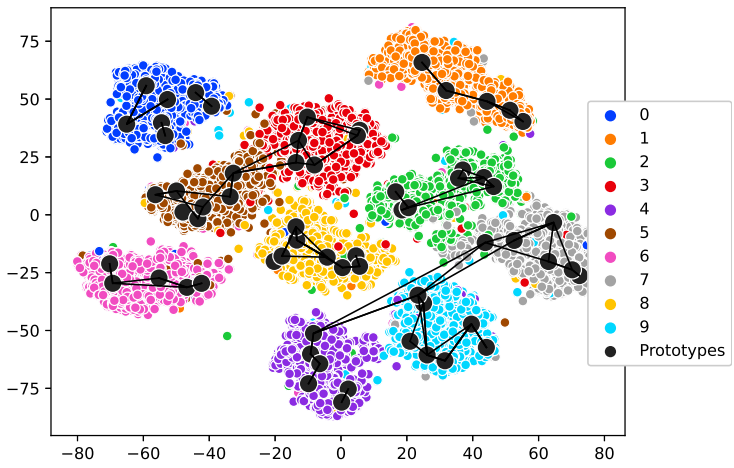


Figure 31: t-SNE: MNIST Test Data + DCSOM-RL Prototypes.

different class regions make sense in a semantic perspective, e.g., the number 9 shares similarities with 4 and 7, in which the roundness of number 4 can transform it into a number 9, and vice versa. These interesting results allow us to observe the characteristics of prototypes found and features shared by prototypes of different categories. These results describe the potential of the

developed SOM layer, and it fulfilled its purpose of learning a useful representation by learning prototypes and performing clustering.

5.6 DCSOM-RL IN JDC

In this section, we evaluate the DCSOM-RL in JDC scenario. The JDC learns the feature space and the clustering in an end-to-end step. It is important to remember that JDC is a harder scenario when compared with previous, due to the fact, that the incorporation of the SOM loss function can make the optimization process diverges if it did not being correctly tuned.

Our focus was to evaluate the model preventing the trivial solutions (one cluster representing the median centroid of all data points in the feature space, or a solution with one cluster per input data point) and creating useful clusters in an end-to-end situation. Considering one cluster representing the median centroid can be avoided with prototypes being attractors regions for similar input data and repulsive regions with lower learning rates for input data that is not so close to the winner prototype, e.g., second winners activations. The over-segmentation of clusters can be avoided controlling the a_t parameter, due that this parameter influences the creation of new nodes in the algorithm.

The analysis was done comparing the results of the DCSOM-RL against the competitors in terms of evaluation metrics, and also, analyzing the reconstructed prototypes, and the visualization of the t-SNE space, showing the relationship of the clusters.

We highlight that all compared models used the same number of clusters. However, due to the time-varying feature of SOM-RL, we need to specify just a limit in the number of clusters.

The next scenario, (JDC), is done with a more rigorously set up, so after we run the LHS to get the best parameters, we run for 10 times changing the seed of each experiment to get the mean and variance of the model.

5.6.1 Results

We expanded the previous section experiments conducting more experiments on MNIST handwritten digits (LeCun *et al.*, 1998), and also, Fashion-MNIST article images (Xiao *et al.*, 2017). For all experiments, the same architecture was used and the results were evaluated from both quantitative and qualitative perspectives.

After adjusting the parameters of the model with LHS, we run a comparison with the other models on the standard MNIST and Fashion-MNIST test sets. It is important to remember that in this scenario (JDC), the losses are combined with learning the latent space and the clustering simultaneously. Because of this, it is essential to investigate the prototypes and the latent space representations carefully.

[2]The authors did not provide a standard deviation, and the public source code was not possible to run.

Table 8: Clustering performance of DCSOM-RL and some baselines on MNIST and Fashion-MNIST in terms of Purity and NMI. The values are the means of 10 runs ± the respective standard deviations. Each method used 64 embeddings/clusters, except DCSOM-RL, due to its time-varying structure. Only the DCSOM-RL was evaluated by us.

Method	MNIST		Fashion-MNIST	
	Purity	NMI	Purity	NMI
k-means	0.791 ± 0.005	0.537 ± 0.001	0.703 ± 0.002	0.492 ± 0.001
SOM-VAE	0.868 ± 0.003	0.595 ± 0.002	0.739 ± 0.002	0.520 ± 0.002
DE-SOM^2	0.939	0.657	0.752	0.538
DCSOM-RL	0.884 ± 0.014	0.521 ± 0.007	0.679 ± 0.009	0.404 ± 0.005

5.6.2 Analysis

The analysis was done comparing the results of the DCSOM-RL against the competitors, similar to the previous experimental scenario. We also did a more detailed quantitative and qualitative analysis, as mentioned before. We run the LHS to get the best parameters, followed by 10 runs of the DCSOM-RL, changing the seed of each experiment to get the mean and variance.

5.6.2.1 Quantitative Analysis

The results are shown in Table 8. We found that our method achieves competitive results concerning its competitors. Although we did not get the best results, we built robust representations with meaningful properties, which will be explored qualitatively. Note that the parameter n_{max} is specified for limiting the amount of memory required.

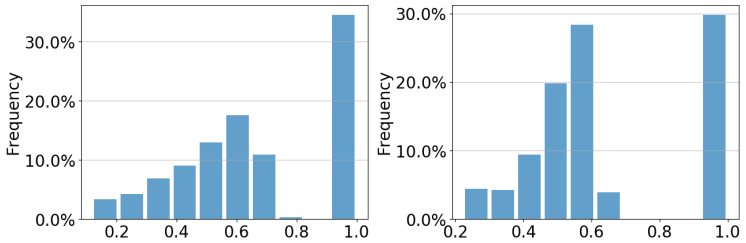
Moreover, other models are originally trained over $10,000$ epochs. However, DCSOM-RL is consistently able to achieve the reported results in less than 30 epochs.

5.6.2.2 Qualitative Analysis

In order to assess whether our model can learn interpretable representations, we analyzed the relevances learned by the model. Note that DCSOM-RL can find samples that live in different subspaces may belong to more than one cluster, the DCSOM-RL is the only model capable of doing it compared to the evaluated baselines. It is a consequence of taking into account different input dimensions' subsets, according to their relevances for what the prototype is trying to represent. A good interpretable behavior happens when the prototype does not represent the raw class itself, but specific characteristics. So, changing its relevant dimensions can degrade part of the prototype without mischaracterizing it. This change can damage the used metrics, but for other tasks that require compositionality, it could be useful, the relevances of the important features may become high, whereas the irrelevant ones become low. In this case, we hypothesize

that changes to these irrelevant features will not degrade the prototype's main characteristics when it is decoded. Analogously, when an important feature is changed, we expect that the prototype loses its properties, and a mischaracterization of the category occurs.

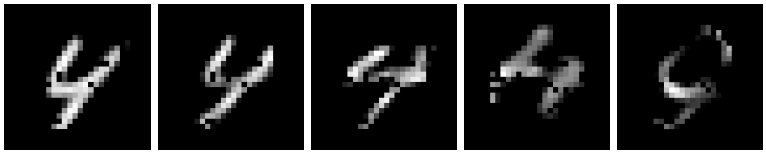
In an effort towards understanding the relevances learned by the model, we analyze the distribution of relevance values found among all prototypes. Figure 32 presents this information. Note that in this particular case, the model considered 35% of the dimensions as highly important (Figure 32a, relevance values close to 1.0 in the x-axis). Fashion-MNIST (Figure 32b) presented 30% of highly important dimensions, and 30% of moderately important dimensions.



(a) MNIST (b) Fashion-MNIST

Figure 32: Histograms of relevances (relevances values on the x-axes) for MNIST and Fashion-MNIST.

To demonstrate the importance of the relevant and irrelevant dimension to the reconstructions, we start from an example prototype (Figure 33a) and disturb the dimensions with low relevance values, such as < 0.3. In this case, we expect that changes in the latent space's dimensions will not have a strong impact on the reconstructed images. Figure 33b shows that the prototype retains its main characteristics. If we expand the perturbation to more relevant dimensions, such as < 0.4 (Figure 33c), we start seeing an impact on the reconstruction. If we push this threshold to dimensions with relevance < 0.5 (Figure 33d), the effect is evident. It degrades, as we expected to see, though it still is recognizable as the digit 4. In opposition, if we disturb the most relevant dimensions for this particular node (> 0.6) the degradation takes place in a sense the node loses its original characteristic (Figure 33e). So, the relevance values found by the model are indeed meaningful. Intuitively, the contrary statement also showed to be true. Low values of relevance mean, in fact, unimportant dimensions for the prototype at hand.

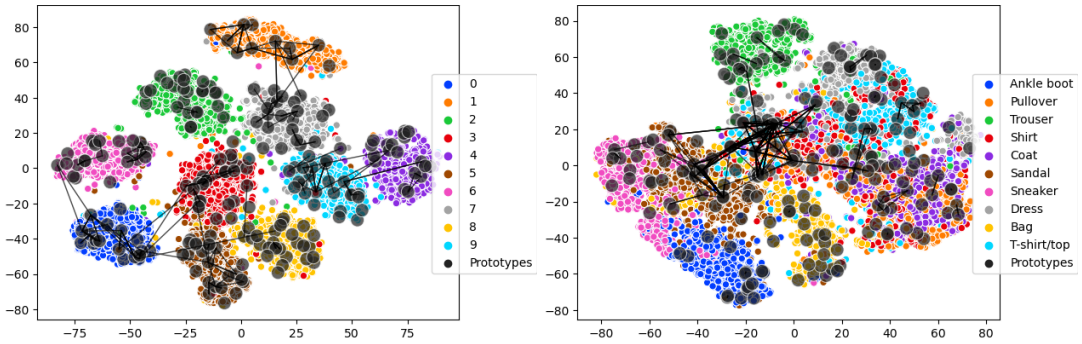


(a) Prototype (b) < 0.3 (c) < 0.4 (d) < 0.5 (e) > 0.6

Figure 33: The decoded prototype on the left (a). Perturbation in dimensions with relevances values less then 0.3 (b), 0.4 (c), and 0.5(d) is inserted. For small values, the prototype does not lose its characteristics. However, when the relevance value increases, the prototype starts to degrade. When only relevant dimensions are changed (e), the node loses its original characteristic.

If we further explore the highest relevance, it is possible to highlight some interesting behaviors. Figure 34 shows the pictures generated by decoding the prototype after varying a single relevant dimension on the latent space. In this particular case, the learned factors relate to an important characteristic that differentiates the numbers 0 and 8, which is the constraint in the middle. If we decrease, it becomes a zero, if we increase, it becomes an eight, in between it is similar to a 3. The proximity between these digit categories can also be observed in Figure 35a. So, the model succeeds in representing specific characteristics of images on its relevant dimensions.

Figure 34: Latent factors learned on MNIST: Transition from 0 to 3, and to 8 obtained by changing only relevant features in the latent space.



(a) MNIST Test Data (b) Fashion-MNIST Test Data

Figure 35: t-SNE for MNIST and Fashion-MNIST Test Data alongside decoded DCSOM-RL Prototypes.

To illustrate the topological structure in the latent space, we present the t-SNE projection (Maaten & Hinton, 2008) of the encoded samples alongside the prototypes found by the model (Figure 35). Notice that, for the MNIST dataset, the model was able to create at least one cluster for each class, connecting more densely prototypes of a same class, and the connections between nodes in regions of different classes usually make sense in a semantic perspective (e.g., the number 9 shares similarities with 4 and 7, the number 7 with 1). For the Fashion-MNIST dataset, we observed some relations between Sandal and Sneaker, Pullover and Coat, but more investigations need to be done to make its feature space more disentangled. These results reveal interesting characteristics of the datasets, expressed in the prototypes found, such as features shared by prototypes of different categories.

With this, we conclude the analysis of the proposed models. In this section, we validated our model for the DC field, more specifically, the JDC scenario, which is an essential factor to

bring powerful models such as LARFDSSOM to the DL in an end-to-end training way. In the next section, we remark some points that are important about the relevance of the work and also discuss some points that still need improvements to bring this approach to the state-of-art.

6 FINAL CONSIDERATIONS

In this thesis, we presented SOM-based models to the context of DL. The proposed DCSOM-RL presents a model combining AE with a customized SOM (SOM-RL). It presents a novel way to deal with high dimensional data, such as images and sound, while learns representations more suitable for clustering in latent spaces. The prototypes identified represent frequent variations observed in the input data. For instance, the different ways to represent the input is caught by the prototypes. It can also bring insights about similarities between different categories or feature representations and which dimensions of the latent space capture then in relation to the prototypical representations found. The neighborhood established leads to smoother regions of transition between categories in the latent space. The work is also the first to use the CE metric for the UVOC tasks, considering its importance for the subspace clustering penalty.

In the current chapter, we draw the final considerations of this Thesis. First, Section 6.1 introduces some analysis of the proposed models discussing for each scenario if the goals of the work were reached. Second, in Section 6.2, we address the main contributions of this Thesis. Third, Section 6.3 describes its limitations. Fourth, Section 6.4 proposes some applications of the models in different scenarios. Finally, in Section 6.5, we present future works that could improve and expand our work.

6.1 ANALYSIS OF THE PROPOSED MODELS

In the first scenario, we focused on UVOC, covering image description methods. For the first time, LARFDSSOM algorithm was evaluated in this challenging field, obtaining the best CE results. We performed a brief image description evaluation, comparing SIFT descriptors combined with several standard interest point detectors, and GoogLeNet. This scenario contributed to our main objective to understand and showed that SOM-based models are compatible with the representations produced by DL models. The main drawback of this scenario was the usage of transfer-learning to extract the features of a pre-trained model, so it was required a pre-trained classifier, which needed labeling for its training.

In the second scenario, we developed the SOM-RL, a SOM layer implemented in PyTorch, with time-varying structure and relevance learning that can process data in batches, to perform training without labels, in a fully unsupervised way. First, we pre-train the AE, and second, we extract the features of the AE and use it on the SOM-RL. This scenario was essential to evaluate the SOM-RL layer in a fully unsupervised context. The main drawback of this scenario was the two steps needed to train the approach because it is not efficient and we also had to specify the number of epochs to pre-train the AE before training SOM-RL. We achieved competitive results compared to DE-SOM and SOM-VAE, but it was an initial study that needs many variables to be considered. In our scenario, the autoencoder was pre-trained with 10x fewer epochs than the number of epochs of DE-SOM and yet it was capable of converging in the trained datasets,

showing the fast convergence of the model. These results showed that the layer called SOM-RL reached our second objective to develop a SOM deep learning layer.

The third scenario was the most ambitious compared to the previous ones. In this scenario, we increased the framework and trained the model end-to-end in a fully unsupervised way. The DCSOM-RL supports complex data and can be used to train deep learning architectures with the same objectives of previous scenarios, but it can also deal with our third objective to guide the organization of the latent space in an end-to-end training fully unsupervised way composed of a SOM and an AE. A question that comes up from the results is why the DCSOM-RL does not improves quantitative results in relation to competitors. We believe that the lack of reproducibility of the experimental setup of DE-SOM model using the same data preprocessing and training hyperparameters may have directly impacted the comparison. Examining the qualitative results, we observed those good prototypes representations and features were obtained that our competitors do not describe in their models. A potential problem of the SOM with DC field is the lack of qualitative analysis of the AE latent space and a detailed analysis of SOM prototypes.

6.2 CONTRIBUTIONS TO SCIENCE

The following list details the main contributions of this work:

- We developed a model capable of backpropagating the unsupervised error of self-organizing maps to artificial neural networks.
- We introduced self-organizing maps based model features for the deep clustering research field.
- We implemented a deep learning layer with self-organizing maps capable of dealing with complex high-dimensional data.
- We proposed a framework capable of studying the latent space of autoencoders.
- We open-source the project for reproduction purposes to stimulate the development of the field.

During the development of this thesis, the article related to the model LARFDSSOM with MSSDC, was published in the Computer Vision and Image Understanding (CVIU), 2019. This work was produced in collaboration with Felipe B. Duque and Aluizio F. Ribeiro. The SOM-RL with MSSDC done in collaboration with Pedro H. Braga was evaluated in a semi-supervision, and it was published at the International Joint Conference on Neural Networks (IJCNN), 2020. To conclude, the SOM-RL with MSSDC was presented on the LatinX workshop at LatinX 2020, and it was also done in collaboration with Pedro H. Braga and Hansenclever F. Bassani. The following list describes the published works:

- **Medeiros**, **H**.**R**., de Oliveira, F.D., Bassani, H.F. and Araujo, A.F., 2019. Dynamic topology and relevance learning SOM-based algorithm for image clustering tasks. Computer Vision and Image Understanding, 179, pp.19-30.
- Braga, P. H. M., **Medeiros**, **H**.**R**. and Bassani, H. F., Deep Categorization with Semi-Supervised Self-Organizing Maps. Proceedings of the International Joint Conference on Neural Networks (IJCNN). IEEE, 2020.
- **Medeiros**, **H**.**R**., Braga, P. H. M. and Bassani, H.F. Deep Clustering Self-Organizing Maps with Relevance Learning. LatinX in AI Research Workshop at the 37th International Conference on Machine Learning (ICML), 2020.

6.3 LIMITATIONS OF THE WORK

Here, we present some limitations of our work. The following list describes some of its limitations:

- It lacks an extensive study of the influence of the architecture in the cluster's compositions.
- A larger study in the generalization capabilities of DCSOM-RL, and it also needs to address more challenging datasets, e.g., ImageNet, CIFAR-10, and CIFAR-100.
- The results maybe do not reflect a real comparison due to the lack of reproducibility of the baselines methods.
- Our models are sensitive to certain parameters like a_t, and it needs to be tuned with a considerable number of trials.
- A robust methodology to study the features disentangle is required to evaluate if the features are disentangled quantitatively and not qualitatively.
- A better way to evaluate the quality of the latent space is needed.

6.4 POSSIBLE APPLICATIONS

The work developed in this thesis can help in DL model explainability, e.g., the activated prototypes are useful for explaining some features response due to a specific input, and we can also make small perturbations in the encoded prototypes in the latent space to understand its topology. The DCSOM-RL can be employed to learn concepts and understand its relationship to objects or words using the relevances. It is important to remember that the DCSOM-RL is a clustering model, so in principle, it can be applied to an extensive list of tasks involving clustering, e.g., image retrieval in cloud databases like in search engines and recommendation system based

on relevances based on the user profile like in media services. The implementation of the SOM-RL as a PyTorch layer makes the layer capable of being used in real-time applications, due to its optimizations to GPU usage, so it could, for example, perform object categorization in computer vision applied to robots.

6.5 FUTURE WORK

Because this Thesis proposed several scenarios, it opens a broad research field with SOM-RL-based models in DC tasks. The closed-loop model was not proposed in this work, but a framework that could use the cluster's information to train the latent space could also be essential to make better disentangle dimensions. Disentangle metrics, such as β-vae score (Higgins *et al.*, 2016), could be employed to perform an online parameter learning to dynamically control the proportion of the loss combination in JDC scenario. We also want to adjust the model for semi-supervised tasks, because it can incorporate *learning vector quantization* (LVQ) skills (Braga & Bassani, 2018, 2019) or a triplet loss (Schroff *et al.*, 2015) to separate better the latent space features to improve cluster. The work done in this thesis could be extended to add more DC variations, e.g., more DL architectures, and it could also be improved to work with online scenarios for example in a real-world computer vision task, where a robot could use the SOM-RL layer in its model to cluster new objects while performing its path-planning. The DCSOM-RL could also have some online analysis of its latent space features using some disentangle metric.

REFERENCES

Aljalbout, E., Golkov, V., Siddiqui, Y., Strobel, M., & Cremers, D. (2018). Clustering with deep learning: Taxonomy and new methods. *arXiv preprint arXiv:1801.07648*.

Baldi, P. (2012). Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of ICML workshop on unsupervised and transfer learning*, 37–49.

Banijamali, E. & Ghodsi, A. (2017). Fast spectral clustering using autoencoders and landmarks. In *International Conference Image Analysis and Recognition*, 380–388.

Bassani, H. F. & Araújo, A. F. (2012). Dimension selective self-organizing maps for clustering high dimensional data. In *The 2012 International Joint Conference on Neural Networks (IJCNN)*, 1–8.

Bassani, H. F. & Araujo, A. F. (2014). Dimension selective self-organizing maps with time-varying structure for subspace and projected clustering. *IEEE transactions on neural networks and learning systems*, 26(3):458–471.

Beeferman, D. & Berger, A. (2000). Agglomerative clustering of a search engine query log. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, 407–416.

Begum, M. & Karray, F. (2011). Visual attention for robotic cognition: a survey. *IEEE Transactions on Autonomous Mental Development*, 3(1):92–105.

Bengio, Y. *et al.* (2009). Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127.

Bengio, Y., Lamblin, P., Popovici, D., & Larochelle, H. (2007). Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, 153–160.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. springer.

Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. *The Journal of Machine Learning Research*, 3:993–1022.

Braga, P. H. & Bassani, H. F. (2018). A semi-supervised self-organizing map for clustering and classification. In *2018 International Joint Conference on Neural Networks (IJCNN)*, 1–8.

Braga, P. H. & Bassani, H. F. (2019). A semi-supervised self-organizing map with adaptive local thresholds. In *2019 International Joint Conference on Neural Networks (IJCNN)*, 1–8.

Caron, M., Bojanowski, P., Joulin, A., & Douze, M. (2018). Deep clustering for unsupervised learning of visual features. In *The European Conference on Computer Vision (ECCV)*.

Cirera, J., Carino, J. A., Zurita, D., & Ortega, J. A. (2020). A data-driven-based industrial refrigeration optimization method considering demand forecasting. *Processes*, 8(5):617.

Cover, T. M. & Thomas, J. A. (1991). Entropy, relative entropy and mutual information. *Elements of information theory*, 2:1–55.

da Silva, L. E. B. & Wunsch, D. C. (2017). An information-theoretic-cluster visualization for self-organizing maps. *IEEE transactions on neural networks and learning systems*, 29(6):2595–2613.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, 248–255.

Deng, L. & Yu, D. (2014). Deep learning: methods and applications. *Foundations and Trends® in Signal Processing*, 7(3–4):197–387.

Erhan, D., Bengio, Y., Courville, A., Manzagol, P.-A., Vincent, P., & Bengio, S. (2010). Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(Feb):625–660.

Erhan, D., Szegedy, C., Toshev, A., & Anguelov, D. (2014). Scalable object detection using deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2147–2154.

Farooq, F., Ahmed, J., & Zheng, L. (2017). Facial expression recognition using hybrid features and self-organizing maps. In *2017 IEEE International Conference on Multimedia and Expo (ICME)*, 409–414.

Forest, F., Lebbah, M., Azzag, H., & Lacaille, J. (2019). Deep architectures for joint clustering and visualization with self-organizing maps. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 105–116.

Fortuin, V., Hüser, M., Locatello, F., Strathmann, H., & Rätsch, G. (2018). Som-vae: Interpretable discrete representation learning on time series. *arXiv preprint arXiv:1806.02199*.

Fritzke, B. (1994). Growing cell structures—a self-organizing network for unsupervised and supervised learning. *Neural networks*, 7(9):1441–1460.

Fritzke, B. (1995). A growing neural gas network learns topologies. In *Advances in neural information processing systems*, 625–632.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.

Graves, A., Mohamed, A.-r., & Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, 6645–6649.

Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7):1645–1660.

Haykin, S. (1994). *Neural networks: a comprehensive foundation*. Prentice Hall PTR.

Helton, J. C., Davis, F., & Johnson, J. D. (2005). A comparison of uncertainty and sensitivity analysis results obtained with random and latin hypercube sampling. *Reliability Engineering & System Safety*, 89(3):305–330.

Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., & Lerchner, A. (2016). beta-vae: Learning basic visual concepts with a constrained variational framework. *International Conference on Learning Representations*.

Hinton, G. E. (2009). Deep belief networks. *Scholarpedia*, 4(5):5947.

Hinton, G. E. (2012). A practical guide to training restricted boltzmann machines. In *Neural networks: Tricks of the trade*, 599–619. Springer.

Hinton, G. E., Osindero, S., & Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554.

Huang, P., Huang, Y., Wang, W., & Wang, L. (2014). Deep embedding network for clustering. In *2014 22nd International conference on pattern recognition*, 1532–1537.

Kalchbrenner, N., Grefenstette, E., & Blunsom, P. (2014). A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*.

Kim, Y. (2014). Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.

Kingma, D. P. & Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Kinnunen, T., Sidoroff, I., Tuononen, M., & Fränti, P. (2011). Comparison of clustering methods: A case study of text-independent speaker modeling. *Pattern Recognition Letters*, 32(13):1604–1617.

Kohonen, T. (1990). The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105.

Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97.

Lawrence, S., Giles, C. L., Tsoi, A. C., & Back, A. D. (1997). Face recognition: A convolutional neural-network approach. *IEEE transactions on neural networks*, 8(1):98–113.

LeCun, Y. (1998). The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Lee, H., Pham, P., Largman, Y., & Ng, A. Y. (2009). Unsupervised feature learning for audio classification using convolutional deep belief networks. In *Advances in neural information processing systems*, 1096–1104.

Li, Z., Tang, Y., & He, Y. (2018). Unsupervised disentangled representation learning with analogical relations. *arXiv preprint arXiv:1804.09502*.

Liang, M. & Hu, X. (2015). Recurrent convolutional neural network for object recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 3367–3375.

Lokesh, S., Kumar, P. M., Devi, M. R., Parthasarathy, P., & Gokulnath, C. (2019). An automatic tamil speech recognition system by using bidirectional recurrent neural network with self-organizing map. *Neural Computing and Applications*, 31(5):1521–1531.

López-Sastre, R. J. (2016). Unsupervised robust feature-based partition ensembling to discover categories. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 114–122.

Maaten, L. v. d. & Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605.

Marsland, S., Shapiro, J., & Nehmzow, U. (2002). A self-organising network that grows when required. *Neural networks*, 15(8-9):1041–1058.

McKay, M. D., Beckman, R. J., & Conover, W. J. (2000). A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 42(1):55–61.

McLoughlin, I., Zhang, H., Xie, Z., Song, Y., & Xiao, W. (2015). Robust sound event classification using deep neural networks. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(3):540–552.

Medeiros, H. R., de Oliveira, F. D., Bassani, H. F., & Araujo, A. F. (2019). Dynamic topology and relevance learning som-based algorithm for image clustering tasks. *Computer Vision and Image Understanding*, 179:19–30.

Min, E., Guo, X., Liu, Q., Zhang, G., Cui, J., & Long, J. (2018). A survey of clustering with deep learning: From the perspective of network architecture. *IEEE Access*, 6:39501–39514.

Müller, E., Günnemann, S., Assent, I., & Seidl, T. (2009). Evaluating clustering in subspace projections of high dimensional data. *Proceedings of the VLDB Endowment*, 2(1):1270–1281.

Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press.

Nemirovski, A. S. & Yudin, D. B. (1978). Cesari convergence of the gradient method of approximating saddle points of convex-concave functions. In *Doklady Akademii Nauk*, 239(5):1056–1059.

Nutakki, G. C., Abdollahi, B., Sun, W., & Nasraoui, O. (2019). An introduction to deep clustering. In *Clustering Methods for Big Data Analytics*, 73–89. Springer.

Patrikainen, A. & Meila, M. (2006). Comparing subspace clusterings. *IEEE Transactions on Knowledge and Data Engineering*, 18(7):902–916.

Peng, X., Feng, J., Xiao, S., Lu, J., Yi, Z., & Yan, S. (2017). Deep sparse subspace clustering. *arXiv preprint arXiv:1709.08374*.

Redmon, J. & Farhadi, A. (2018). Yolov3: An incremental improvement. *CoRR*, abs/1804.02767.

Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, 91–99.

Rezende, D. J., Mohamed, S., & Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., *et al.* (2015). Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252.

Saito, S., Wei, L., Hu, L., Nagano, K., & Li, H. (2017). Photorealistic facial texture inference using deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 5144–5153.

Schroff, F., Kalenichenko, D., & Philbin, J. (2015). Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 815–823.

Strehl, A. & Ghosh, J. (2002). Cluster ensembles—a knowledge reuse framework for combining multiple partitions. *Journal of machine learning research*, 3(Dec):583–617.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1–9.

Treisman, A. M. & Gelade, G. (1980). A feature-integration theory of attention. *Cognitive psychology*, 12(1):97–136.

Tuytelaars, T., Lampert, C. H., Blaschko, M. B., & Buntine, W. (2010). Unsupervised object discovery: A comparison. *International journal of computer vision*, 88(2):284–302.

Vidal, R. (2011). Subspace clustering. *IEEE Signal Processing Magazine*, 28(2):52–68.

Vincent, P., Larochelle, H., Bengio, Y., & Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, 1096–1103.

Xiao, H., Rasul, K., & Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.

Xie, J., Girshick, R., & Farhadi, A. (2016). Unsupervised deep embedding for clustering analysis. In *International conference on machine learning*, 478–487.

Yang, B., Fu, X., Sidiropoulos, N. D., & Hong, M. (2017). Towards k-means-friendly spaces: Simultaneous deep learning and clustering. In *international conference on machine learning*, 3861–3870.

Zeiler, M. D. & Fergus, R. (2014). Visualizing and understanding convolutional networks. In *European conference on computer vision*, 818–833.