TESIS DE LA UNIVERSIDAD DE ZARAGOZA

2014

25

Enrique Pelayo Campillos

Magnitude Sensitive Competitive Neural Networks

Departamento
Ingeniería Electrónica y Comunicaciones

Director/es
Buldain Pérez, Julio David
Orrite Uruñuela, Carlos

ISSN 2254-7606

Prensas de la Universidad Universidad Zaragoza

Universidad Zaragoza
1542

Tesis Doctoral

MAGNITUDE SENSITIVE COMPETITIVE NEURAL NETWORKS

Autor

Enrique Pelayo Campillos

Director/es

Buldain Pérez, Julio David
Orrite Uruñuela, Carlos

UNIVERSIDAD DE ZARAGOZA

Ingeniería Electrónica y Comunicaciones

2014

Magnitude Sensitive Competitive Neural Networks

By

Enrique Pelayo Campillos

Dissertation presented to the Department of Departamento de Ingeniería Electrónica y Comunicacionesin partial fulfillment of the requirements for the degree of

Doctor of Philosophiae

PhD Advisors:

David Buldain Pérez

Carlos Orrite Uruñuela (Co-director)

At

Escuela de Ingeniería y Arquitectura

Universidad de Zaragoza, 2014

Abstract

Competitive Learning is a kind of unsupervised learning commonly used to find a solution to the approach of Vector Quantization (VQ) task. VQ tries to generate a reduced representation of the vector-data distribution used in its training process.

There are many Competitive Learning algorithms, but most of them share the property of distributing their centroids over the data with a density proportional to the probability density function of the data. That results in modeling representations that tend to concentrate centroids in the densest areas of the data distribution.

However, some quantization applications may require an inverse relation between the codewords density and the data density, as in many biological systems. It is not just that, sometimes it is required a codeword distribution independent of data density.

In this Thesis it is presented a set of Neural Networks called *Magnitude Sensitive Competitive Neural Networks (MSCNNs)*. They are a set of neural competitive learning algorithms, that include a magnitude term as a modulation factor of the distance used for the unit competition. As other competitive methods, MSCNNs perform a vector quantization of the data, however the magnitude factor leads centroids to represent with higher detail any desired zones, defined by this factor. This distribution is improved with the use of independent learning factor for each unit, calculated form the magnitude, as it will be explained in the description of the algorithms.

Two MSCNNs neural networks are developed: *MSCL (Magnitude Sensitive Competitive Learning)* a hard competitive algorithm, and *MS-SOM (Magnitude Sensitive Self Organizing Maps)* a soft competitive algorithm, that shares with MSCL its capability of allocating centroids in data-distribution zones according to an arbitrary magnitude, but additionally preserves topological information of the data. Both algorithms are analysed, as well as their 'masked' versions (implementation that may be trained with data samples

containing invalid components).

MS-INITis a new codebook initialization algorithm, developed as a generalization of the known KKZ and K-Means++, but taking into account the magnitude.

MSCL, MS-SOM and MS-INIT(in their different implementations) were compared with other vector quantization approaches in several examples of interpolation, image color quantization, surface modelling, classification, and some simple artificial examples. Additionally it is presented a new image compression algorithm, *MSIC (Magnitude Sensitive Image Compression)* that make use of these new algorithms, and achieves a level of compression different along the image according to the user defined magnitude.

Results show that the new MSCNNs are more versatile than other competitive learning algorithms in certain tasks, and present a clear improvement in vector quantization over them when data is weighted by a magnitude that marks the 'interest' of each sample.

Resumen

El aprendizaje competitivo (Competitive Learning, CL) es un tipo de aprendizaje no supervisado que se usa habitualmente para encontrar una solución al problema de cuantificación vectorial (Vector Quantization, VQ). En este tipo de tareas el problema consiste en conseguir una representación reducida de la distribución de datos de entrada usada en el proceso de entrenamiento.

Hay muchos tipos de algoritmos de competitivo, pero la mayoría de ellos comparten la propiedad de distribuir sus centroides sobre los datos con una densidad proporcional a la función de densidad de probabilidad de los datos. Como resultado, los centroides se concentran en las zonas más densas de la distribución de datos.

Sin embargo, algunas aplicaciones de cuantización de datos requieren una relación inversa entre la densidad de los datos y de los centroides, como sucede en muchos sistemas biológicos. Y no solo eso, algunas veces se requiere una distribución que sea totalmente independiente de los datos.

En esta Tesis se presentan un conjunto de redes neuronales llamadas: *Magnitude Sensitive Competitive Neural Networks (MSCNNs)*. Se trata de un conjunto de algoritmos de Competitive Learning que incluyen un término de magnitud como un factor de modulación de la distancia usada en la competición. Al igual que otros métodos competitivos, MSCNNs realizan la cuantización vectorial de los datos, pero el término de magnitud guía el entrenamiento de los centroides de modo que se representan con alto detalle las zonas deseadas, definidas por la magnitud. Esta distribución de las unidades se mejora con el uso de un coeficiente de aprendizaje independiente para cada unidad, calculado a partir de la magnitud, tal y como se explicará a lo largo de la descripción de los algoritmos.

Se han desarrollado dos redes neuronales de tipo MSCNN: *MSCL (Magnitude Sensitive Competitive Learning)*, algoritmo de tipo 'hard competitive' y *MS-SOM (Magnitude Sensi-*

tive Self Organizing Maps), un algoritmo donde la competición es de tipo 'soft competitive', que comparte con MSCL su capacidad de ubicar centroides en determinadas zonas de la distribución de datos de acuerdo a una magnitud arbitraria, pero adicionalmente, consigue mantener mantener en el mapa de la red neuronal las propiedades topológicas de los datos. En la Tesis he desarrollado y estudiado en profundidad ambos algoritmos, y sus versiones 'enmascaradas' (Estas versiones permiten que el entrenamiento se realice con muestras que incluyen alguna componente inválida).

También se muestra un nuevo algoritmo de inicialización del codebook, MS-INIT, que es una generalización de los conocidos KKZ and K-Means++, pero teniendo en cuenta la magnitud.

MSCL, MS-SOM y MS-INIT(en sus distintas implementaciones) se comparan con otros algoritmos de cuantización vectorial en diversos ejemplos de interpolación, reducción de color, modelado de superficies, clasificación, y varios ejemplos sencillos de demostración. Además se introduce un nuevo algoritmo de compresión de imágenes, *MSIC (Magnitude Sensitive Image Compression)*, que hace uso de los algoritmos mencionados previamente, y que consigue una compresión de la imagen variable según una magnitud definida por el usuario.

Los resultados muestran que las nuevas redes neuronales MSCNNs son más versátiles que otros algoritmos de aprendizaje competitivo, y presentan una clara mejora en cuantización vectorial sobre ellos cuando el dato está sopesado por una magnitud que indica el 'interés' de cada muestra.

Dedicatoria

Por tu bondad y sacrificio me inspiraste a ser mejor y ahora puedo decir que esta tesis lleva mucho de tí, gracias por estar siempre a mi lado, Gloria.

Agradecimientos

Es difícil entender la importancia de los agradecimientos de una tesis doctoral hasta que no se ha terminado. En ese momento te das cuenta de cuánto tienes que agradecer a tanta gente. Intentaré resumir en unas líneas la gratitud que siento a todas las personas que han estado presentes durante esa etapa, haciendo posible que hoy deje de ser un sueño para pasar a ser una realidad.

En primer lugar agradezco al Dr. David Buldain, director de esta tesis y amigo mio tras tanto tiempo trabajando juntos, por su orientación y apoyo durante toda la investigación. Sus ideas me han estimulado, y ha sabido centrarme hasta conseguir que de ideas haya llegado a trabajos científicos con el suficiente rigor académico.

Agradezco también al Dr. Carlos Orrite, codirector de la tesis, por su disponibilidad y colaboración en este trabajo. El me abrió puertas del grupo de investigación del *CV-lab*, como participante del proyecto SOCIAL-BEHAVE, dándome la oportunidad de tener una visión más amplia del mundo de la investigación y descubrir cuánto me motiva. Además tuvo un seguimiento muy cercano a mi labor de investigación especialmente en mis primeras etapas de la tesis.

También deseo agradecer el apoyo moral recibido por mis amigos de Ingeniería, Daniel, Jesús y Jose Antonio. Aunque se dedican a otras áreas de conocimiento alejadas de la presente tesis, comprenden su dificultad y desde el principio me han apoyado para seguir adelante con la misma. Por su puesto no me olvido de Jose María, Carlos y otros compañeros compañeros de trabajo, como Antonio y Carlos, mis jefes, que me han apoyado en mi formación. Aunque normalmente ellos vean la I+D "desde la barrera", saben lo complejo que es investigar, especialmente si se hace en ratos libres como yo.

Por último, pero no menos importante, deseo agradecer a mi familia su apoyo. Primero a mis padres Enrique y Angelines, por infundirme el espíritu científico que está detrás de

todos mis estudios. A mis hijas Carmen y María por haberme perdonado el que les haya robado tantas horas de juegos juntos. Y finalmente a Gloria, la persona que más me ha ayudado a sacar adelante la tesis. Sin su sacrificio y su apoyo emocional durante todo éste tiempo hubiese sido totalmente imposible.

Muchas gracias a todos.

El autor también desea agradecer al I3A, al Departamento de Ingeniería Electrónica y Comunicaciones de la Escuela de Ingeniería y Arquitectura de la Univ. de Zaragoza y al CDTI por su confianza. Este trabajo ha sido parcialmente financiado con la subvención concedida por el Ministerio de Ciencia e Innovación y el fondo Social Europeo (TIN2010-20177).

Contents

List of Figures

List of Tables

Part I

INTRODUCTION

Chapter 1

Introduction

1.1 Introduction and motivation

Competitive Learning is a kind of unsupervised learning commonly used to find a solution to the Vector Quantization (VQ) task. VQ tries to generate a reduced representation of the vector-data distribution used in its training process. The resulting weight-vectors (prototypes) of the units are denominated centroids, or codewords, and the set of centroids of the neurons constitute the codebook of the VQ representation. Many practical applications can benefit from VQ methods.

Well known Competitive Learning approaches are K-means ([49], [48]), Frequency Sensitive Competitive Learning (FSCL) [1], Self-Organizing Maps (SOM) [38], Neural Gas (NG) [50], Elastic Net (EN) [24] and Generative Topographic Mapping (GTM) [11]. All these methods distribute their centroids over the data distributions with a density proportional to the probability density function of the data. This type of codification is optimal from the point of view of maximizing the Shannon's Information-Theory entropy for the use of codewords in a transmission task.

On the contrary, some applications may require an inverse relation between codewords density and data density. It has been demonstrated that in biological systems, unusual stimuli are differentiated with high precision, whereas frequent stimuli are distinguished only in a rough manner. This effect is usually called as 'perceptual magnet' effect [39], [40]. Some of the variants of the previously mentioned methods achieve a codebook representation inversely proportional to data density thorough '*magnification control*'. Next chapter will explain how it works.

However, none of these methods solve the issue of dataset modeling with units following

a probability distribution different from density. For instance, in biological systems unusual stimuli are not only dependant on its frequency of appearance but also on its context. In an image, the position, orientation, size or color of an object may be relevant to define saliency. An elephant in the Savanna is not surprising, but it is if you see an elephant in a street. Or in the original image if the elephant is magenta. Or an elephant floating in the sky.

In artificial datasets similar problem arises: In satellite image quantization it may be interesting to compress images but giving more importance to certain areas of the image, depending on the image characteristics (i.e. blue color if the goal is information on water) or regular shapes if the goal is getting information on buildings.

Another example is economic analysis. In the temporal evolution of financial markets it is usual that certain shapes arise, shapes which economists associate with some important economic facts. It is interesting to process economical data modelling in detail these significant data samples.

Other possible issue that is not completely solved using common CL algorithms is Anomaly Detection in complex industrial processes. These processes usually imply multivariate (high dimensional) data samples that may be abnormal depending on the circumstances. A trained operator is able to deal with the whole information to define if the product from a certain process is normal or abnormal (i.e. it has the required quality). However it may be difficult to retrieve all the necessary information so an automatic unsupervised learning method could process correctly the novelty information intrinsic to the dataset.

Conventional Competitive Learning methods do not give a good data model representation in these cases. That is the reason that motivated this Thesis. In the Thesis we present a group of new competitive neural networks, *Magnitude Sensitive Competitive Neural Networks (MSCNN)*, that have the property of distributing centroids in data-distribution zones according to an arbitrary magnitude calculated or obtained locally for each unit, opening not only the possibility of allocating codewords in function of data density, but also in function of any other user-defined target magnitude.

These algorithms are: *Magnitude Sensitive Competitive Learning (MSCL)*, *Magnitude Sensitive Self Organizing Maps (MS-SOM)*, and their masked versions. Additionally it is presented a new method for initializing the codebook, method that also takes the magnitude into account (MS-INIT).

Throughout the Thesis, several examples of the use of these algorithms have been shown, including some real application examples in the field of computer vision. I decided

to use this kind of applications to demonstrate the advantages of the new algorithms against other conventional methods for two reasons:

- Results are very visual and self-explanatory.
- I belong to the *Computer Vision Lab* from the University of Zaragoza (`http://i3a.unizar.es/en/content/cvlab`).

1.2 Thesis organization

This Thesis is structured in four parts with some appendices containing complementary materials to the main core of the Thesis.

Part I is composed of Chapters 1 and 2. It is an introductory block where some of the most important Competitive Neural Networks are revised with special attention to those most related with the new ones developed in this Thesis.

Part II includes Chapters 3 to 6. It contents the description of the new competitive learning algorithms developed within the Thesis.

Part III (chapters 7 to 9) shows the use of the new algorithms in three different computer vision related tasks: color quantization, image compression and 3D surface modelling.

Finally, last part includes a concluding chapter with a review of the Thesis achievements and explains future open research areas. Appendices contain the description of additional materials, including the *MS Toolbox*, a Matlab toolbox [52] to work with all the algorithms previously explained.

1.3 Notational conventions

This section covers the general style of notational and mathematical expressions used in this thesis. Mostly matrix operations are used.

Matrices are denoted as upper-case boldface letters, e.g. $\mathbf{X}$. When referring to the i-th row vector of a matrix, the lower-case bold letter of the matrix letter is used, and a subscript is written to denote the row index, respectively. For example, matrix $\mathbf{X}$'s i-th row vector would be written as $\mathbf{x}_i$. When referring to a single element of matrix $\mathbf{X}$ in row i and column j, it is written as a lower-case italic letter with subscripts (with or without comma), e.g. $\mathbf{x}_{ij}$ or $\mathbf{x}_{i,j}$

Vectors are denoted as lower-case boldface letters. The elements of a vector, e.g. $\mathbf{v}$, are written with a subscript to denote its index in the vector: $\mathbf{v} = (v_1, v_2, \ldots, v_k)$.

Symbol	Meaning
$\|\cdot\|$	Euclidean norm
$\mathcal{S}$	Set
$\|\mathcal{S}\|$	Cardinality of $\mathcal{S}$, i.e. the number of elements of a set
$\mathbf{x}$	A vector
x_i	i-th element of vector $\mathbf{x}$
$\mathbf{X}$	A matrix
$\mathbf{x}_i$	i-th row of matrix $\mathbf{X}$
$x_{ij}/x_{i,j}$	Element (i,j) of matrix $\mathbf{X}$

Table 1.1: Summary of notational conventions.

Notation for matrix and vector operation is the universally accepted. We only want to mark that in the chapter 5, it is necessary to use element-wise vector products, which are denoted as '$\circ$' .

The different models described in this Thesis share several architectural properties and some specific names are used to denote units or sample data.

Each network consists of a set of M units:

$$\mathcal{M} = \{u_1, u_2, \ldots, u_M\} \tag{1.1}$$

with unit weights $\mathbf{w}_m \in \mathbb{R}^D$ (corresponding to unit u_m) indicating its position or receptive field center in input space.

The n-dimensional input signals are assumed to be generated from a finite training data set of length equal to N:

$$\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}, \mathbf{x}_i \in \mathbb{R}^D \tag{1.2}$$

During training, the sample presented to the neural network at time t is denoted as $\mathbf{x}(t)$.

Given one input signal $\mathbf{x}(t)$, the winner j (also called Best Matching Unit, or BMU) among the units in $\mathcal{M}$ is defined as the unit with the nearest reference vector.

Usually in competitive neural networks, distance is measured by euclidean distance. Then j equals:

$$j = \underset{u_m \in \mathcal{M}}{\operatorname{argmin}}(\|\mathbf{x}(t) - \mathbf{w}_m(t)\|) \tag{1.3}$$

However, in the Magnitude Sensitive Competitive Neural Networks developed in this

Thesis it is not used this definition of Best Matching Unit, because the magnitude must be taken into account in the competition.

For convenience we define the *Voronoi Region* of a unit u_m as the Voronoi region of its reference vector:

$$V_m = \{\mathbf{x} \in \mathbb{R}^D \mid BMU(\mathbf{x}) = u_m\} \tag{1.4}$$

In the case of a input data set $\mathcal{X}$ we denote for a unit u_m with the term *Voronoi Set* the subset $\mathcal{R}_m$ of $\mathcal{X}$ for which u_m is the winner

$$\mathcal{R}_m = \{\mathbf{x} \in \mathcal{X} \mid BMU(\mathbf{x}) = u_m\} \tag{1.5}$$

Additionally we define Quantization Error of one sample $\mathbf{x}$(t) with BMU j as:

$$Q_{err}(t, j) = \|\mathbf{x}(t) - \mathbf{w}_j(t)\| \tag{1.6}$$

The mean of its squared value for all the units is the Mean Squared Error:

$$MSE(\mathcal{X}; \mathcal{M}) = \frac{1}{\mid \mathcal{X} \mid} \cdot \sum_{\substack{u_m \in \mathcal{M} \\ \mathbf{x} \in \mathcal{R}_m}} \|\mathbf{x} - \mathbf{w}_m\|^2 \tag{1.7}$$

Chapter 2

Competitive Learning Neural Networks

2.1 Introduction

Unsupervised learning concerns the problem of trying to find hidden structures in unlabelled data. Competitive Learning is a form of unsupervised learning based on artificial neural networks, in which nodes compete for the right to respond to a subset of the input data. A common goal of those algorithms is to distribute a certain number of vectors (prototypes or centroids) in the input data space. The distribution of these vectors should reflect (in one of several possible ways) the probability distribution of the input signals which in general is not given explicitly but only through sample vectors.

Competitive Learning Neural Networks usually contain an unique layer of neurons with unit weights $\mathbf{w}_i \in \mathbb{R}^D$ which is commonly known as 'competitive layer'. The training process is usually divided in two steps:

1. Competition phase: For every input vector $\mathbf{x}$, neurons 'compete' with each other to see which one of them is the most similar to that particular input vector (that unit is called 'winner').

2. Weights updating phase: In this step, one or more of the units move towards the input sample.

There are two paradigms of Competitive Learning algorithms depending on which of the units are updated. *Hard competitive learning* (a.k.a. winner-take-all learning) comprises methods where each input signal only determines the adaptation of one unit, the winner. A general problem occurring with hard competitive learning is the possible existence of 'dead units'. Another problem of hard competitive learning is that different

random initializations may lead to very different results.

The second paradigm is called *Soft competitive learning* (a.k.a. winner-take-more learning). In this case not only the winner but also some other units move towards the sample input. Unit adaptation is different depending on similarity of each unit to the winner unit or the input data, being always greater for winner.

Training can be obtained by performing either batch or on-line update. In batch methods all possible input signals (which must come from a finite set in this case) are evaluated first, before any adaptation is done. This is iterated a number of times. On-line methods, on the other hand, perform an update step directly after each input signal. The possibility of this on-line training, in conjunction with the simplicity of the method is an advantage over other unsupervised learning algorithms.

CL neural networks are used for different applications depending on the goal of the used CL method . Main applications are:

1. **Vector quantization**: Vector Quantization is a classical quantization technique which allows the modelling of probability of the density of a dataset by the distribution of a set prototype vectors. Input dataset is divided into groups having approximately the same number of points closest to them. Each group is represented by a prototype vector (codeword).

   The goal in Vector Quantization is the minimization of the expected quantization (or distortion) error between that codeword and the input vectors that it represent. Correspondingly, in the case of a finite data set $\mathcal{D}$ the Mean Squared Error has to be minimized:

$$MSE(\mathcal{X};\mathcal{M}) = \frac{1}{\mid \mathcal{X} \mid} \cdot \sum_{\substack{u_m \in \mathcal{M} \\ \mathbf{x} \in \mathcal{R}_m}} \|\mathbf{x} - \mathbf{w}_m\|^2 \tag{2.1}$$

2. **Entropy maximization**: If we interpret the generation of an input signal and the subsequent mapping onto the nearest unit in $\mathcal{M}$ as random experiment which assigns a value $u \in \mathcal{M}$ to the random variable X, the Shannon entropy (a measure of the information content in the codification), is defined as:

$$H(X) = -\sum_{u_m \in \mathcal{M}} p(u_m) \log p(u_m) \tag{2.2}$$

   Here, $p(u_m)$ is the probability mass function of outcome u_m, that is, the probability

that input signal is assigned to unit u_m:

$$p(u_m) = \frac{\mid \mathcal{R}_m \mid}{\mid \mathcal{X} \mid} \tag{2.3}$$

Therefore, the entropy becomes:

$$H(X) = -\sum_{u_m \in \mathcal{M}} \left(\frac{\mid \mathcal{R}_m \mid}{\mid \mathcal{X} \mid} \right) \log \left(\frac{\mid \mathcal{R}_m \mid}{\mid \mathcal{X} \mid} \right) \tag{2.4}$$

When the goal of the CL algorithm is entropy maximization, ensuring that all the codewords are used with equal frequency, the probability $p(u)$ tends to the following value as the neural network is trained:

$$p(u_m) = \frac{1}{\mid \mathcal{X} \mid} \tag{2.5}$$

3. **Clustering**: Another possible application of CL algorithms is clustering, where a partition of the dataset into subgroups or clusters is sought, such that data samples in the same group (called cluster) are more similar (in some sense or another) to each other than to those in other groups (clusters). Clustering is widely used for pattern recognition, feature extraction, image segmentation, function approximation, and data mining.

4. **Feature mapping**: Sometimes it is necessary to produce a low-dimensional representation of the input space of the training samples in such a way, that some similarity relations present in the original data are still present after mapping. This process has been denoted feature mapping and can be useful for data visualization. It requires a neural network with fixed dimensionality such as SOM (it will be explained later).

5. **Novelty detection**. The goal is the determination whether or not a unknown data sample $\mathbf{x}$ is well represented by a previously trained CL neural network . The sample is said to be 'known' if the distance from the sample to its BMU ($\mathbf{w}_j$) in the CL neural network is less than a pre defined threshold Th:

$$\|\mathbf{x} - \mathbf{w}_j\| < Th \tag{2.6}$$

Otherwise it is a 'new' sample. The value of Th is usually defined from the mean and standard deviation of the quantitation error reached during the neural network

training.

In this chapter we will analyse some of the existing Competitive Learning algorithms. The remainder of this chapter is organized as follows: Next section describes the basic Competitive Learning Algorithm. Section 2.3 explains some of the most common CL algorithms. Finally, last section describes some other algorithms related to the *Magnitude Sensitive Competitive Neural Networks* described in this Thesis.

2.2 Basic Competitive Learning Algorithm

Here we present the basic Competitive Learning Algorithm in an online training mode. It follows these steps:

1. The codebook set is initialized to contain M units, with codewords $\mathbf{w}_i$ initialized with samples randomly selected from the dataset $\mathcal{D}$:

$$\mathcal{M} = \{u_1, u_2, \ldots, u_M\} \tag{2.7}$$

2. Selection at random of an input signal $\mathbf{x}(t)$ from the dataset $\mathcal{X}$.

3. Determination of winner unit u_j as:

$$j = \underset{u_m \in \mathcal{M}}{\operatorname{argmin}}(\|\mathbf{x}(t) - \mathbf{w}_m(t)\|) \tag{2.8}$$

4. Adaptation the codeword of the winner towards the input sample:

$$\mathbf{w}_j(t+1) = \mathbf{w}_j(t) + \alpha(t) \cdot (\mathbf{x}(t) - \mathbf{w}_j(t)) \tag{2.9}$$

   where $\alpha(t)$ is the learning factor. It takes a constant value or decreases during training.

5. Step 2 is repeated until a stopping condition is reached (e.g. a pre-defined number of iterations).

This CL algorithm is the basis of other algorithm that will be explained in this chapter. Its implementation in batch mode is called LBG (or generalized Lloyd) algorithm.

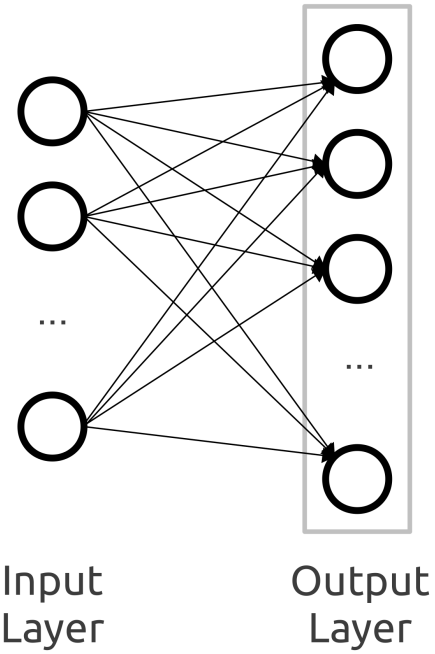


Figure 2.1: Competitive neural network architecture

2.3 Usual Competitive Learning Algorithms

Here we will describe some of the most extended competitive learning algorithms.

2.3.1 K-Means

K-means algorithm ([49]) is the basic CL algorithm, using different values for α for each unit:

$$\alpha_m = \frac{1}{t_m} \tag{2.10}$$

In this case, t_m stands for the number of input signals for which unit u_m has been winner so far. Doing so, codeword $\mathbf{w}_m(t)$ is always the exact arithmetic mean of the input signals it has been winner for. The number of units is K, what originates its name.

k-means is rather easy to implement and apply even on large data sets, and therefore it has been widely used in clustering. As such, it has been successfully used in various topics, ranging from market segmentation, computer vision, geostatistics and astronomy.

The algorithm has two main drawbacks:

1. It requires to define the number of codewords, what may be a problem in clustering

when it is unknown the number of clusters in advance.

2. K-means is sensitive to initialization as it is a Hard Learning algorithm.

2.3.2 Neural Gas

This algorithm is named 'neural gas' because of the dynamics of the feature vectors during the adaptation process, which distribute themselves like a gas within the data space. The algorithm follows the Soft Competing paradigm, what means that all of the units move through the input sample with a step size decreasing with the increase of the distance order.

This adaptation of the whole codebook yields to a robust convergence, but on the other hand processing is more time consuming than K-means.

Neural Gas follows same steps than the basic CL algorithm, but it changes in steps 3 and 4:

3. The distance order of the feature vectors to the given data vector $\mathbf{x}$ is determined. i_0 denotes the index of the closest feature vector ($\mathbf{w}_{i_0}$), i_1 the index of the second closest feature vector ($\mathbf{w}_{i_1}$), ..., and i_{N-1} the index of the feature vector most distant to $\mathbf{x}$.

4. In the adaptation step, each codeword $\mathbf{w}_{i_k}(k = 0, \ldots, M-1)$ is adapted according to:

$$\mathbf{w}_{i_k}(t+1) = \mathbf{w}_{i_k}(t) + \alpha(t) \cdot e^{-k/\lambda(t)} \cdot (\mathbf{x}(t) - \mathbf{w}_{i_k}(t)) \tag{2.11}$$

   Both $\alpha(t)$, and $\lambda(t)$, the so-called neighborhood range, are reduced with increasing time t.

Growing neural gas (Fritzke, 1994b, 1995a) is a variant of this algorithm where the number of units is increased during training (beginning with very few units) until the global quantization error measures lower a pre-defined value. This version of Neural Gas has the advantage that it does not require to define in advance the number of units as a parameter.

2.3.3 Self-Organizing Feature Map

Self-Organizing Feature Map (also called SOM) is a is a type of CL neural network that has the capability of mapping the D-dimensional input space (being D arbitrarily large) to a lower dimension structure (usually called *map*), while preserving the topological properties of the input space. This makes it possible to get a representation of the data which may be used for visualization purposes (in this case it uses a 2D or 3D representation).

The SOM is also a Soft Competive Algorithm that follows the basic CL algorithm excepting in step 3, as all of the units (not only the winner) are updated.

3. For each unit $u_i \in \mathcal{M}$:

$$\mathbf{w}_m(t+1) = \mathbf{w}_m(t) + \alpha(t) \cdot h_{mj}(t) \cdot (\mathbf{x}(t) - \mathbf{w}_m(t)) \tag{2.12}$$

   In this equation, $h_{mj}(t)$ is the neighborhood function that depends on the lattice distance between the BMU (u_j) and unit u_m. A Gaussian function is a common choice for this neighborhood function.

The advantage of SOM against other CL algorithms is the ordered topological structure of neurons in data space. SOM may be considered a nonlinear generalization of Principal Components Analysis (PCA) [84]. It has been shown, using both artificial and real geophysical data, that SOM has many advantages over it ([46], [47]).

Due of these advantages, SOM has been widely used in several machine learning applications. They include image and video processing; density or spectrum profile modeling; text/document mining and management systems; gene expression data analysis and discovery; and high dimensional data visualization.

Being such a popular unsupervised learning algorithm, it has several variants. Some of them are:

- Time adaptive self-organizing map (TASOM) network ([72]) is an extension of the basic SOM. TASOM employs adaptive learning rates and neighborhood functions. It also includes a scaling parameter to make the network invariant to scaling, translation and rotation of the input space. The TASOM and its variants have been used in several applications including adaptive clustering, multilevel thresholding, input space approximation, and active contour modeling.

- Growing self-organizing map (GSOM, [3]) is a growing variant of the self-organizing map. GSOM was developed to address the issue of identifying a suitable map size for the SOM. It starts with a minimal number of nodes and grows new nodes during training. By using a value called the spread factor, the data analyst has the ability to control the growth of the GSOM.

- There are also several temporal extensions of the SOM to have into account the effect of the temporal occurrence of the input samples. Temporal Kohonen Map (TKM,

[15]) and Recurrent Self-Organizing Map (RSOM, [78]), incorporate leaky integrator memory to preserve the temporal context of the input signals.

- Other variants are related with the number of neural networks used simultaneously as in the case of *Hierarchical SOMs*, where, at some stage, one of the SOMs receives as inputs the outputs of another SOM.

2.4 Most related methods

There are several methods highly related to MSCL or MS-SOM. Here we explain some of them.

2.4.1 Frequency Sensitive Competitive Learning

The most similar one is the FSCL method [1], that basically follows the same structure as MSCL if we use the winning frequency of the units as magnitude.

The FSCL introduces during the competition phase a parameter, named the relative winning frequency or 'conscience'. The centres chance to win the competition is directly proportional to the relative winning frequency. The learning rate of the frequent winners is reduced, as their chance to win the competition does. By this additional 'conscience' term, it circumvents the 'dead units' problem usually present in the k-means algorithm.

FSCL uses the basic implementation of CL, with a modified criterion for selection of the best-matching unit by adding the 'conscience' term F, that depends on the number of hits that every unit has received up to the moment (w_m). Therefore, step 3 is different:

$$j = \underset{u_m \in \mathcal{M}}{\operatorname{argmin}}(F(w_m) \cdot \|\mathbf{x}(t) - \mathbf{w}_m(t)\|) \tag{2.13}$$

where usually $F(w_m)$ takes an exponential form:

$$F(w_m) = (w_m)^\gamma \tag{2.14}$$

Some successful applications of the FSCL algorithm are feature extraction [13] and image compression [14]. Some variants of FSCL has been developed for clustering tasks, as the Rival Penalized Competitive Learning algorithm RPCL [12]. This algorithm rewards the winning center and penalizes with a de-learning rate the second winner, named rival.

2.4.2 Energy Based Competitive Learning

Energy Based Competitive Learning [82] is an algorithm that addresses the three important issues associated with competitive learning clustering. Auto-initialization is achieved by extracting samples of high energy to form a core point set, whereby connected components are obtained as initial clusters. To adapt to clusters of different size and sparsity, a novel competition mechanism, uses for each prototype a definition of energy multiplied by distance, to select a winner prototype. For eliminating the disturbance caused by outliers, adaptive learning rate based on samples' energy is proposed to update the winner.

The reason of using energy is to select a number of prototypes to cluster the data distributions optimally according to its density, so that it differs considerably from MSCL, that is formulated as a magnitude-oriented VQ method.

2.4.3 K-Harmonic Means and Weighted K-means

Two methods based on K-means, K-Harmonic Means [86] and Weighted K-means [37], use the modulation of the distance by factors as MSCL does with magnitude. The first method replaces the minimum distance from a data point to the centroids, used in K-means, by the Harmonic Averages of the distances from the data point to all centroids. The main goal of this algorithm is that the K-means becomes less sensitive to the initialization of the centroids. However authors claim that the method significantly improves the quality of clustering results comparing with both K-Means and Expectation Maximization [22].

The Weighted K-means uses the same structure of K-Harmonic Means, introducing membership function for the centroids, but replacing the Harmonic Averages of the distances by weights obtained from a density-biased reservoir sampling algorithm, that represents the density of the original data points. Both methods are oriented to specific tasks by means of their weighted distance, but basically do not define an open method to any desired target in the VQ processing as MSCL does.

2.4.4 Magnification control

After training a CL neural network, the achieved weight vector density $\rho(\mathbf{w})$ is in relation to the data density $P(\mathcal{D})$ following:

$$P(\mathcal{X}) \propto \rho(\mathbf{w})^{\alpha} \tag{2.15}$$

The exponent α is called magnification exponent or magnification factor. Controlling this factor it is possible to modify the magnification properties of the vector quantizer as

it may be required in different application tasks.

Magnification and its control is related to biological phenomena like the 'perceptual magnet effect'. In 1992, Kuhl, introduced this phenomenon, which demonstrated that as a second language is acquired, the brain gradually groups sounds according to their similarity with phonemes in the native language. That was due to the fact that rarely occurring stimuli are differentiated with high precision whereas frequent stimuli are distinguished only in a rough manner ([39]; [40]).

It is a kind of attention-based learning with inverted magnification, where rarely occurring input samples are emphasized by an increased learning gain.

This effect is also beneficial in technical systems. For instance in remote-sensing image analysis, seldomly found ground cover classes should be detected, whereas usual (frequent) classes with broad variance should be suppressed ([54], [80]). Also anomaly detection in industrial processes or intrusion detection in computer security systems requires that abnormal situations receive more importance during training.

Villmann presents in ([81]) a general framework to control the magnification achieved by SOM or NG neural networks. This method works by following one of these learning schemes:

1. *Localized learning*: Introduction of a multiplicative factor by a local learning rate $\alpha_m = \alpha(\mathbf{w}_m)$ that depends on the stimulus density P at the position of their weight vectors $\mathbf{w}_m$ via
$$\langle \alpha_m \rangle = \alpha_0 \cdot P(\mathbf{w}_m)^\gamma , \tag{2.16}$$
where the brackets $\langle\rangle$ denote the average in time.

2. *Winner-relaxing learning*: Introduction of winner relaxing by adding a winner-enhancing (relaxing) term R:
$$\mathbf{w}_j(t+1) = \mathbf{w}_j(t) + \alpha(t) \cdot (\mathbf{x}(t) - \mathbf{w}_j(t)) + R \tag{2.17}$$

3. *Concave-convex learning*: Scaling of the learning shift by powers ξ in the factor $(\mathbf{x} - \mathbf{w}_i)^\xi$:
$$\mathbf{w}_j(t+1) = \mathbf{w}_j(t) + \alpha(t) \cdot (\mathbf{x}(t) - \mathbf{w}_j(t))^\xi \tag{2.18}$$

Part II

ALGORITHMS

Chapter 3

MSCL algorithm

3.1 Introduction

3.1.1 Proposed approach

MSCL is a parallelizable algorithm, as other neural networks, that admits on-line data training and follows the general *Competitive Learning* steps:

Selection the winner prototype . Given an input data vector, the competitive units compete each other to select the winner neuron comparing their prototypes with the input. This Best Matching Unit (BMU) is selected in MSCL as the one that minimizes the product of the magnitude (provided with data or calculated from a user-defined function and assigned to each unit) and the distance of the unit prototypes to the input data vector. This procedure differs from other usual competitive algorithms where the BMU is determined only by distance. The MSCL competition is implemented by a two-step competition: global and local competitions, as will be explained in section 3.2.

Updating the winner and magnitude . Winner's weights are adjusted iteratively for each training sample, with a learning factor specific for each unit, and forced to decay with training time. Concurrently, magnitude at each unit is also updated.

3.1.2 Magnitude

MSCL algorithm uses a user-defined magnitude function, $MF()$, that acts as an extra information for the network, forcing neurons to represent with more detail those zones of

data space with higher magnitude values. The idea behind the use of the magnitude as a weighting factor in the competition by distance is that, in case of a sample placed at equal distance from two competing units, the winner will be the unit with lower magnitude value. One key point of the method is the appropriate definition of the magnitude function for the desired VQ task. Output of this function is always a positive scalar, and it takes the general form:

$$MF : [\mathbf{w}_m(t), < m >, \mathcal{X}, \langle args \rangle] \rightarrow \mathbb{R}^+ \tag{3.1}$$

Here we refer with $\langle args \rangle$ as optional arguments for the function. As they are optional they may have different sizes. For the sake of clarity we will avoid these arguments in the rest of the Thesis just for clarity.

There exists mainly two situations depending on the data dependency of this function $MF()$:

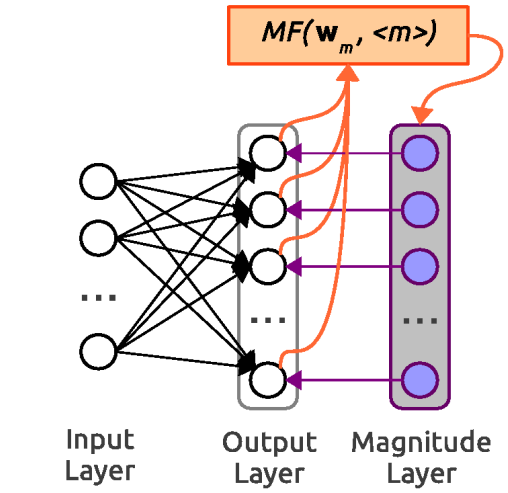
1. When magnitude function depends on neuron data, $MF(\mathbf{w}_m(t), < m >)$, we define for each neuron u_m an internal variable, $mu_m(t)$. This variable for each unit forms an additional magnitude layer which is used during local competition and is of equal size than the output layer or map. It is calculated as:

$$mu_m(t) = MF(\mathbf{w}_m(t), < m >) \tag{3.2}$$

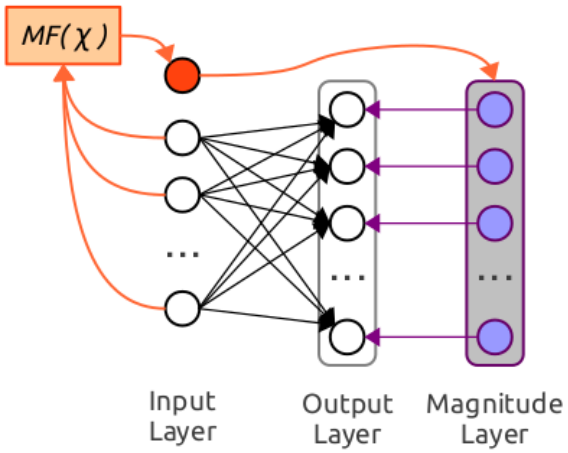
   With $< m >$ we represent any variable related to unit u_m, for instance samples in its Voronoi region V_m, or neigboring units of unit u_m. From now on, we will use the term "magnitude map" when magnitude is calculated from the units, see Figure 3.1(a).

2. When magnitude is determined exclusively from input data, we use $MF(\mathcal{X})$ and define a magnitude vector, $\mathbf{mx}$, that is included as an extra input for the neurons of the map. This value at each sample is used to calculate the value of the magnitude associated to each unit. From now on, we will use the term "magnitude vector" when magnitude is an external value associated to the data patterns, see Figure 3.1(b). Then, $mx(t)$ is equal to:

$$mx(t) = MF(\mathbf{x}(t)) \tag{3.3}$$



(a) Using $MF(\mathbf{w}_m(t), < m >)$



(b) Using $MF(\mathcal{X})$

Figure 3.1: Architecture of the MSCL neural network using two alternative definitions of the magnitude function: a) as a magnitude map evaluated into units, or b) as an external magnitude vector associated to data patterns.

3.1.3 Chapter description

The remainder of this chapter is organized as follows. Next section describes in more detail the *Magnitude Sensitive Competitive Learning (MSCL)* method. Section 3.3 analyses the influence of its parameters in its behaviour.

Section 3.4 shows the comparison of the method with well known methods used in VQ tasks, as FSCL, Fuzzy C-means clustering (FCM), Neural Gas (NG), K-Means and Self-Organizing Maps (SOM). Two application examples are proposed to show the different results that MSCL generates, as an oriented method by the magnitude function, compared with the results of these known methods. The first proposed application is to model a toy problem with 5000 data samples in three gaussian distributions, comparing the final results with three evaluation measures for the VQ representations generated in the methods without supervision. The second application is a simple problem designed to show how can MSCL focus units in zones of high variability when interpolating data series, compared with FSCL.

3.2 The MSCL algorithm

Next subsections describe the algorithm in an iterative updating schedule, whose flowchart is shown in figure 3.2, and also an updating schedule in batch mode, where units are adjusted after a presentation of a number of samples or after the complete presentation of the whole dataset

3.2.1 Online implementation of the MSCL algorithm

Initialization

M unit weights are initialized with data inputs randomly selected from the dataset, and their initial value of its magnitude is equal to the magnitude function at these samples.

$$\mathbf{w}_m(0) = \mathbf{x}(m) \quad m = 1 \dots M \tag{3.4}$$

Then, unit magnitude might be initialized by the magnitude function ($MF()$) depending only on unit parameters (equation 3.5a), or alternatively by the value of the magnitude at the selected sample data for each unit (eq. 3.5b):

$$mu_m(0) = MF(\mathbf{w}_m(0), < m >) \tag{3.5a}$$

$$mu_m(0) = mx(m) \tag{3.5b}$$

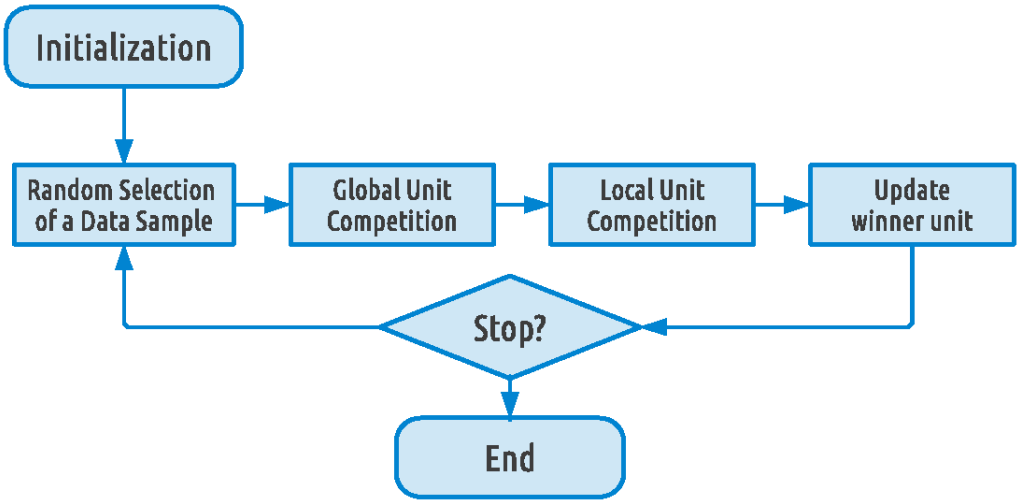


Figure 3.2: MSCL flowchart

where mx is the magnitude of the sample $\mathbf{x}(m)$. The initial accumulated magnitude of unit u_m, is set to:

$$macc_m(0) = mu_m(0) \tag{3.6}$$

Random selection of data samples

A sample data $\mathbf{x}(t) = (x_1, .., x_n)(t) \in \mathbb{R}^D$ is randomly selected at time t from the dataset $\mathcal{X}$. This process will be repeated until every data has been presented to the MSCL neural network. It is recommended to retrain the neural network with the whole dataset several cycles, along C input data presentations (iterations), to make results independent of data-presentation ordering.

Global unit competition

K units with minimum distance from their weights to the input data vector are selected as winners in this first step. These units form the $\mathcal{S}$ set ($\mathcal{S} \subset \mathcal{M}$):

$$\begin{gathered} \mathcal{S} = \{u_{s1}, u_{s2}, ..., u_{sK}\} \\ \|\mathbf{x}(t) - \mathbf{w}_s(t)\| < \|\mathbf{x}(t) - \mathbf{w}_m(t)\| \; \forall u_m \notin \mathcal{S} \wedge u_s \in \mathcal{S} \; . \end{gathered} \tag{3.7}$$

Local unit competition

In the second step, winner unit with index j is selected from units belonging to $\mathcal{S}$ as the one that minimizes the product of its magnitude value as in equation 3.8a (or the accumulated magnitude, eq. 3.8b) with the distance of its weights to input data vector, following one of these equations

$$j = \underset{u_s \in \mathcal{S}}{\operatorname{argmin}}(mu_s(t)^\gamma \cdot \|\mathbf{x}(t) - \mathbf{w}_s(t)\|) \ , \text{ or} \tag{3.8a}$$

$$j = \underset{u_s \in \mathcal{S}}{\operatorname{argmin}}(macc_s(t)^\gamma \cdot \|\mathbf{x}(t) - \mathbf{w}_s(t)\|) \ , \tag{3.8b}$$

being $macc_s(t)$ the accumulated magnitude of unit u_s calculated by the equation 3.10

The use of mu in local competition is more adequate than $macc$ when the goal of training is Q_{err} reduction while $macc$ is better to reduce the entropy. γ is an exponential factor to modulate the strength of the magnitude during the competition.

Winner update

For all units in the map, weights and magnitude are adjusted iteratively for each training sample, following ($m = 1 \dots M$):

$$mf(t) = \begin{cases} mx(t), & \text{if used a magnitude vector } (\mathbf{mx}). \\ mu_j(t), & \text{otherwise.} \end{cases} \tag{3.9}$$

$$macc_j(t+1) = macc_j(t) + mf(t) \tag{3.10}$$

$$\alpha_m(t) = \left(\frac{mf(t)}{macc_m(t+1)} \right)^\beta \tag{3.11}$$

$$\mathbf{w}_j(t+1) = \mathbf{w}_j(t) + \alpha_m(t)\,(\mathbf{x}(t) - \mathbf{w}_j(t)) \tag{3.12}$$

where α is the learning rate calculated for the winner and forced to decay with the magnitude accumulation and β is a scalar value between 0 and 1. Using this definition, when β is equal to one, the value of each unit's weights become the weighted running mean of the input data samples belonging to its Voronoi region.

Magnitude update

Only winner's magnitude is adjusted for each training sample, following:

$$mu_j(t+1) = \begin{cases} mu_j(t) + \alpha_m(t)\left(mx(t) - mu_j(t)\right), & \text{if } \mathbf{mx} \text{ is used.} \\ MF(\mathbf{w}_j(t+1), < j >), & \text{otherwise.} \end{cases} \tag{3.13}$$

Stopping condition

Training finish when a termination condition is reached: it may be the situation when all data samples has been presented to the MSCL neural network along certain number of cycles (if a limited number of samples is used), or the condition of low mean change in unit weights, or any other function that could measure the training stabilization.

3.2.2 Batch implementation of the MSCL algorithm

Next subsections describe the algorithm in an batch updating schedule, where units are adjusted after a presentation of a limited number of samples at each epoch or after the complete presentation of the whole dataset. Here we name t_{ep} the epoch number.

Initialization

Initialization is done in the same way than in online MSCL implementation.

Random selection of data samples

We select a data set $\mathcal{X}_{ep} \subset \mathcal{X}$ formed of N_{ep} randomly selected samples from the dataset:

$$\mathcal{X}_{ep} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{N_{ep}}\}, \mathbf{x}_i \in \mathbb{R}^D \tag{3.14}$$

Global unit competition

For each sample in $\mathcal{X}_{ep}$, K units with minimum distance from their weights to the input data vector are selected as winners in this first step. These units form the $\mathcal{S}(t)$ set, corresponding to input sample $\mathbf{x}$(t) ($\mathcal{S}(t) \subset \mathcal{M}$):

$$\begin{aligned} \mathcal{S}(t) &= \{u_{s1}, u_{s2}, ..., u_{sK}\} \\ \|\mathbf{x}(t) - \mathbf{w}_s(t)\| &< \|\mathbf{x}(t) - \mathbf{w}_m(t)\| \; \forall u_m \notin \mathcal{S}(t) \wedge u_s \in \mathcal{S}(t) \\ t &= 1 \dots N_{ep}. \end{aligned} \tag{3.15}$$

Local unit competition

Once again, for each sample in $\mathcal{X}_{ep}$, best matching unit j is selected from units belonging to S as the one that minimizes the product of its magnitude value as in equation 3.16a (or the accumulated magnitude, eq. 3.16b) by the distance of its weights to input data vector, following one of these equations:

$$j = \underset{u_s \in \mathcal{S}(t_{ep})}{\operatorname{argmin}} \left(mu_s(t)^\gamma \cdot \|\mathbf{x}(t) - \mathbf{w}_s(t)\| \right) \text{ , or} \tag{3.16a}$$

$$j = \underset{u_s \in \mathcal{S}(t_{ep})}{\operatorname{argmin}} \left(macc_s(t)^\gamma \cdot \|\mathbf{x}(t) - \mathbf{w}_s(t)\| \right) \text{ .} \tag{3.16b}$$

Winner update

First, the magnitude of input sample at time $t, mf(t)$, is calculated through equation 3.9. Then for each unit u_m, we calculate the weighted arithmetic mean ($\mathbf{xep}_m$) of samples belonging to $\mathcal{X}_{ep}$ within its Voronoi Region $\mathcal{R}_m$, using the value of the magnitude $mf(t)$ as the value for weighting the sample $\mathbf{x}(t)$:

$$\mathbf{xep}_m = \frac{\sum\limits_{k \in \mathcal{R}_m} mf(k) \cdot \mathbf{x}(k)}{\sum\limits_{k \in \mathcal{R}_m} mf(k)} \tag{3.17}$$

being $\mathcal{R}_m = \{\mathbf{x} \in \mathcal{X}_{ep} \mid j = u_m\}$.

The estimated increase of magnitude in the epoch for that unit is:

$$mep_m = \sum_{k \in \mathcal{R}_m} mf(k) \tag{3.18}$$

These values are used for calculating the learning rate α_m, and update the accumulated magnitude and weight of each unit:

$$macc_m(t_{ep}+1) = macc_m(t_{ep}) + mep_m \tag{3.19}$$

$$\alpha_m = \left(\frac{mep_m}{macc_m(t_{ep}+1)}\right)^{\beta} \tag{3.20}$$

$$\mathbf{w}_m(t_{ep}+1) = \mathbf{w}_m(t_{ep}) + \alpha_m\left(\mathbf{xep}_m - \mathbf{w}_m(t_{ep})\right) \tag{3.21}$$

where β is a scalar value between 0 and 1.

Magnitude update

Each unit u_m updates its magnitude. If this value is given by a value associated to each input sample, magnitude at each unit becomes the running weighted mean of the magnitude of all data samples at its Voronoi region:

$$mxep_m = \frac{\sum\limits_{k\in\mathcal{R}_m} mx(k)^2}{\sum\limits_{k\in\mathcal{R}_m} mx(k)} \tag{3.22}$$

$$mu_m(t_{ep}+1) = mu_m(t_{ep}) + \alpha_i\left(mxep_m - mu_m(t_{ep})\right) \tag{3.23}$$

Otherwise, it is calculated directly with the magnitude function:

$$mu_m(t_{ep}+1) = MF(\mathbf{w}_m(t_{ep}+1), < M >) \tag{3.24}$$

Stopping condition

Training finish when a termination condition is reached, as in the case of online MSCL implementation. Otherwise the process will be repeated from step 2 every epoch until every data has been presented to the MSCL neural network.

3.3 Algorithm analysis

3.3.1 Resulting Voronoi regions

As mentioned in Chapter1, section 3, Voronoi region of a unit u_m consists of all points in the Euclidean space where this unit is the Best Matching Unit of these points.

When Euclidean distance is used in the competition step in a Competitive Learning algorithm, decision region between adjacent Voronoi regions is formed by hyperplanes. These hyperplanes consist of all the points in the space equidistant to the two nearest unit-prototypes.

In this case,

$$V_m = \{\mathbf{x} \in \mathbb{R}^D \mid \|\mathbf{x} - \mathbf{w}_m\| \leq \|\mathbf{x} - \mathbf{w}_k\| \; \forall u_k \in \mathcal{M}\} \tag{3.25}$$

However, with the magnitude term weighting the distance in a two step competition (as in the MSCL algorithm), the resulting boundaries between Voronoi regions can be more complex and those units with higher magnitude tend to have lower volumes in their Voronoi regions, even if the data density is uniform.

Example of figure 3.3(a) shows the result of training 30 units in a two-dimensional synthetic data set consisting of $P = 5000$ samples generated from a mixture of three Gaussian distributions centered on the points (0,0), (3,4) and (6,0) with covariance matrix [0.1 0; 0 0.1] for all of them. The percentage of samples placed in each cluster is almost the same: 33.3%.

We use a magnitude associated to each sample through the function:

$$MF(\mathbf{x}) = abs\,(x_2) \tag{3.26}$$

As can be appreciated in the figure, magnitude takes null values in the horizontal line with $x_2 = 0$, and takes value 1 in the higher value of the second component of $\mathbf{x}$. This figure, 3.3, shows the Voronoi regions corresponding to Euclidean distance (c) and those corresponding to a magnitude weighted distance (d). Each limit between two neighbouring units are equidistant from both units in the case (c), while it is closer to the unit of lower magnitude in the case (d).

3.3.2 Connections

MSCL is a winner-take-all neural network, therefore each unit forms no structure with its neighbouring units. However, for certain applications it is interesting to know what is the neighborhood of a given unit. To do it we define a set of the neighborhood connections between units of the network as:

$$\mathcal{C} = \mathcal{M} \times \mathcal{M} \tag{3.27}$$

where two units $(u_i, u_j) \subset \mathcal{C}$ are 'connected' if both are first and second BMUs considering

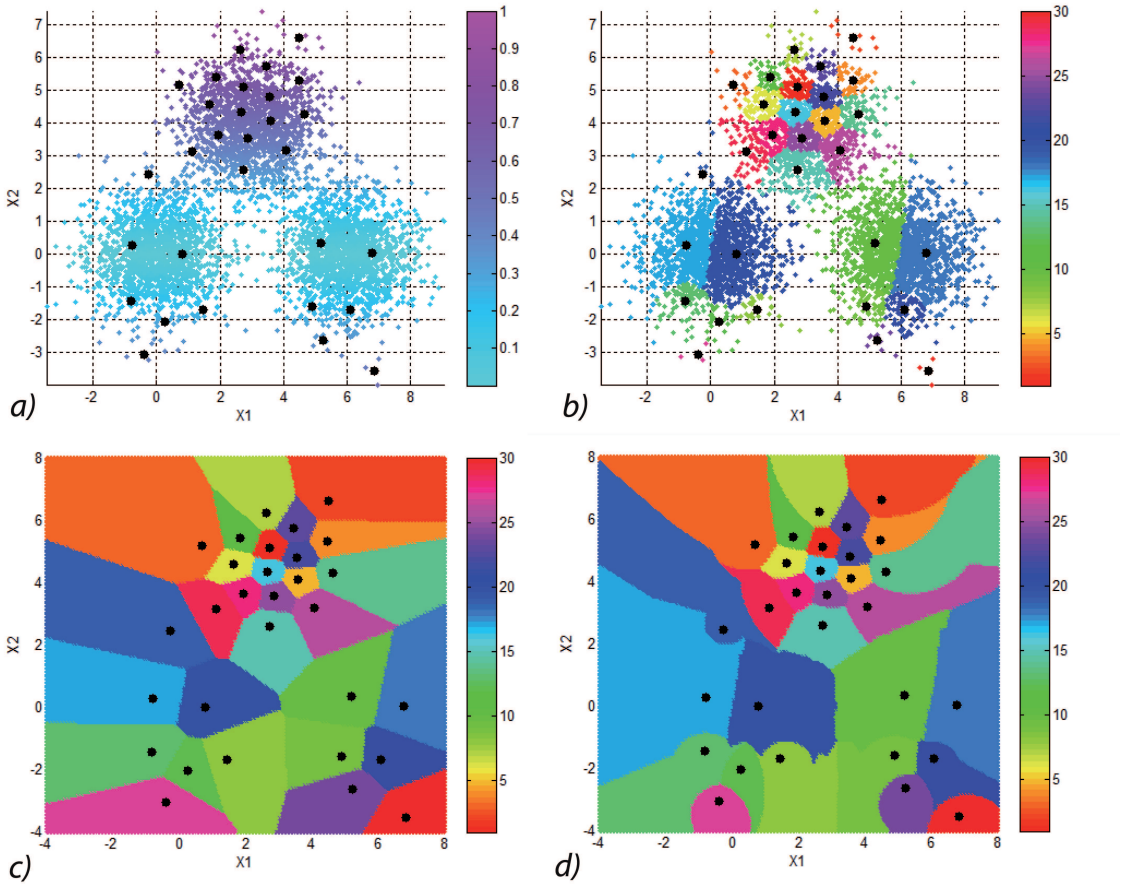


Figure 3.3: Example of Voronoi regions corresponding to 30 units. (a) Centroids and magnitude, abs(x_2). It can be appreciated how the density of centroids increases as x_2 becomes higher, as higher magnitude zones attract more units to be represented. (b) Voronoi sets of each unit: Samples assigned to centroids using distance weighted by magnitude defined in a, (c) Voronoi regions for Euclidean distance, (d) Voronoi regions for distance weighted by magnitude defined in a.

the euclidean distance:

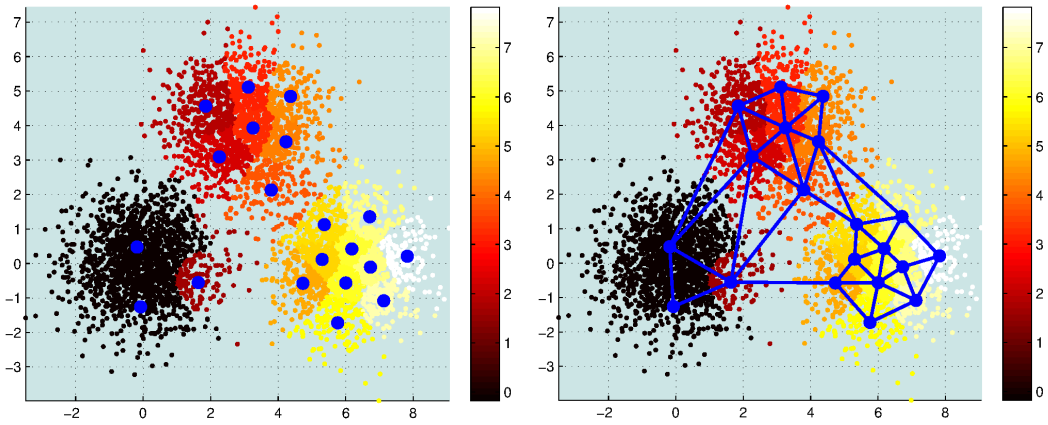
$$i = \underset{u_k \in \mathcal{M}}{\operatorname{argmin}} \left(\|\mathbf{x}(t) - \mathbf{w}_k(t)\| \right) \tag{3.28}$$

$$j = \underset{u_k \in \mathcal{M} \setminus \{u_i\}}{\operatorname{argmin}} \left(\|\mathbf{x}(t) - \mathbf{w}_k(t)\| \right) \tag{3.29}$$

Usually connections set $\mathcal{C}$ is generated once training has finished. This set can also be calculated during the training process. In this case, as units are moving in the data space, connections established during the early stages of training usually does not correspond

with the final ones. Therefore, it becomes necessary to remove old connections. We define that two units $(u_i, u_j) \notin \mathcal{C}$ if they have not been connected during any of the previous $K_{\mathcal{C}}$ input signal presentations, where $K_{\mathcal{C}}$ is an integer that usually corresponds to the total number of data samples.

Figure 3.4 shows the result of training of a MSCL with 20 units (centroids painted in blue) in the same dataset of previous example, but using $abs(x_1)$ as the magnitude. In figure 3.4(a) each sample of the dataset is coloured by the value of the magnitude of its BMU. On the right, it is shown the same figure, but adding the connections between units as it has been explained above.



(a) Trained MSCL (b) Trained MSCL with connections

Figure 3.4: Example of representation of a trained MSCL with connections obtained after training.

3.3.3 Modified quality measures

Two new magnitudes *Weighted Mean Squared Error* and *Weighted Entropy* has been defined to measure the quality of training in a Magnitude Sensitive Competitive Neural Network.

- **Weighted Mean Squared Error**: Weighted Mean Square Error (WMSE) is the weighted mean of the quantization squared error, where weights are the values of the magnitude at each sample (normalized by the sum of magnitudes in all the dataset), and it is calculated in a similar way to [66]:

$$WMSE(\mathcal{X}; \mathcal{M}) = \frac{\sum_{\substack{u_m \in \mathcal{M} \\ \mathbf{x} \in \mathcal{R}_m}} MF(\mathbf{x}) \cdot \|\mathbf{x} - \mathbf{w}_m\|^2}{\sum_{\mathbf{x} \in \mathcal{X}} MF(\mathbf{x})} \tag{3.30}$$

Given a finite data set $\mathcal{X}$, and a defined number of units M, the WMSE value is lower as the distribution of centroids are replicating in more detail the magnitude function.

- **Weighted Entropy**: It is also possible to define a value of Weighted Entropy, where weights are the value of the magnitude at each sample. It also follows 2.3, but using a different definition for the probability $p(u_m)$, that depends on the density of magnitude, instead of the data density.

$$p(u_m) = \frac{\sum\limits_{\mathbf{x}\in\mathcal{R}_m} MF(\mathbf{x})}{\sum\limits_{\mathbf{x}\in\mathcal{X}} MF(\mathbf{x})} \tag{3.31}$$

By dividing by the maximum entropy (obtained when all units have the same value in eq.3.31), the normalized entropy is expressed as:

$$H(X) = -\sum_{u_m\in\mathcal{M}} \left(\frac{\sum\limits_{\mathbf{x}\in\mathcal{R}_m} MF(\mathbf{x})}{\sum\limits_{\mathbf{x}\in\mathcal{X}} MF(\mathbf{x})} \right) \log \left(\frac{\sum\limits_{\mathbf{x}\in\mathcal{R}_m} MF(\mathbf{x})}{\sum\limits_{\mathbf{x}\in\mathcal{X}} MF(\mathbf{x})} \right) / \log\left(\mid \mathcal{M} \mid\right) \tag{3.32}$$

Figure 3.5 shows the evolution of WMSE (black line) and Weighted Entropy (red line) during training of a MSCL with 50 units along 5 cycles of 10 epochs each. Both magnitudes present an abrupt change at the beginning of training and they move towards its final values but more slowly. Dataset $\mathcal{X}$ is the three Gaussian example with magnitude function of equation 3.26.

Table 3.1 shows a comparison of the final values of WMSE and normalized Weighted Entropy, with a MSCL and the basic CL algorithm. Both test are done with different number of units: 25, 100 and 400. As it can be seen in the table, WMSE is always lower

Q measure	*Algorithm*	25	100	400
WMSE	MSCL	0.543	0.290	0.142
	CL	0.585	0.304	0.145
Weighted Entropy	MSLC	0.962	0.947	0.941
	CL	0.885	0.912	0.932

Table 3.1: Comparison of the final values of WMSE and normalized Weighted Entropy in MSCL and a basic CL method, trained with the gaussian dataset and magnitude function of eq. (3.26) for three codebook sizes (25,100 and 400 units).

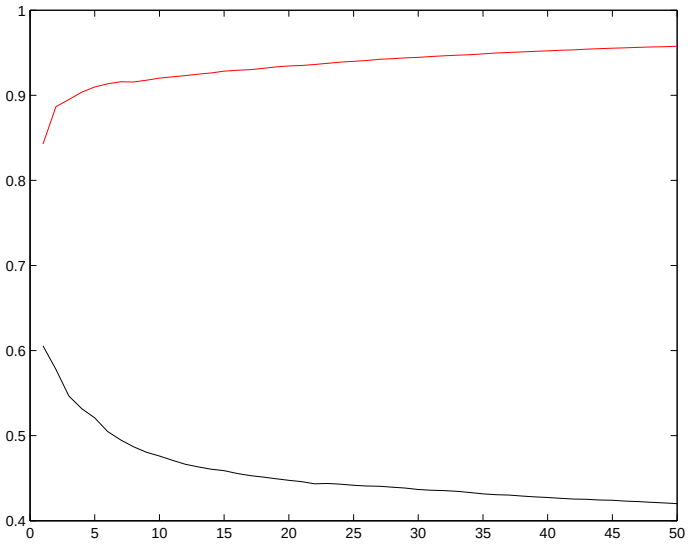


Figure 3.5: Evolution of WMSE (black) and normalized Weighted Entropy (red) during training of a MSCL with 50 units along 5 cycles of 10 epochs each. As expected, WMSE decreases during training, while the normalized Weighted Entropy tends to a constant value of 1.

in MSCL, and the Weighted Entropy is always higher. This means that MSCL is better than simple CL for all the cases independently if the goal is error minimization or entropy maximization.

3.3.4 Effect of alpha

As it has been mentioned in 3.2.1, α is the learning rate, that depends on the magnitude of input samples and is different for each unit. It is assigned a value so that the vector reference $\mathbf{w}_m(t)$ for unit u_m is always the exact arithmetic weighted mean of the input signals it has been the winner. This only happens when β is equal to 1 (Next subsection will explain the influence of β).

If we name $\mathbf{y}(t)$ $(t = 1 \ldots \mid \mathcal{R}_m \mid)$ to the successive inputs where u_m is its BMU, and $my(t)$ the corresponding value of magnitude. The sequence of successive values of $\mathbf{w}_m$,

when equations 3.10 to 3.12 are applied for beta=1, is the following:

$$
\begin{aligned}
\mathbf{w}_m(0) &= \mathbf{y}(0) \\
\mathbf{w}_m(1) &= \mathbf{w}_m(0) + \left(\frac{my(1)}{my(0)+my(1)}\right)\cdot(\mathbf{y}(1)-\mathbf{w}_m(0)) \\
&= \frac{my(0)\cdot\mathbf{y}(0)+my(1)\cdot\mathbf{y}(1)}{my(0)+my(1)} \\
\mathbf{w}_m(2) &= \mathbf{w}_m(1) + \left(\frac{my(2)}{my(0)+my(1)+my(2)}\right)\cdot(\mathbf{y}(2)-\mathbf{w}_m(1)) \\
&= \frac{my(0)\cdot\mathbf{y}(0)+my(1)\cdot\mathbf{y}(1)+my(2)\cdot\mathbf{y}(2)}{my(0)+my(1)+my(2)} \\
&\vdots \\
\mathbf{w}_m(t) &= \mathbf{w}_m(t-1) + \left(\frac{my(t)}{my(0)+my(1)+\ldots+my(t)}\right)\cdot(\mathbf{y}(t)-\mathbf{w}_m(t-1)) \\
&= \frac{\sum\limits_{i=1\cdots t} my(i)\cdot\mathbf{y}(i)}{\sum\limits_{i=1\cdots t} my(i)}
\end{aligned}
\tag{3.33}
$$

Therefore, as desired, the reference vector becomes the arithmetic weighted mean of samples in its Voronoi region. It should be noted that the set of signals $y(t)$ for which a particular unit u_m has been the winner may contain elements which lie outside the final Voronoi region of u_m due to changes in the value of $\mathbf{w}_m$ during training.

3.3.5 Effect of beta

β is an scalar between 0 and 1 (both values are possible) used to add a forgetting factor for unit adaptation during training. As it has been shown in the previous subsection, a value of $\beta = 1$ results in reference vectors that are equal to the arithmetic weighted mean of samples in its Voronoi Set.

However, usually units change substantially during first epochs of training, while they tend to be stationary at last stages. The parameter β is used precisely to enhance the importance of last samples during training (when units are close to its final position). A low value of β (near zero) gives less weight to initial samples when calculating the weighted arithmetic mean of input samples. In the extreme situation, if $\beta = 0$ then $\alpha = 1$ and:

$$\mathbf{w}(t) = \mathbf{x}(t) \tag{3.34}$$

A good selection for β, given the desired final value for α is:

$$\beta = \frac{log(\alpha_{final})}{log\left(\frac{M}{P \cdot C}\right)} \tag{3.35}$$

being C the number of cycles, M the number of units, and P the number of samples.

3.3.6 Effect of gamma

Parameter γ is a positive scalar used to enhance the effect of the magnitude during local competition step. High values make competition dependent only on the value of the magnitude, and values near zero confine the competition to the euclidean distance. Using $\gamma = 0$ means that the competition is done only by distance.

Is important to highlight that for a null value of γ, winner will be the same during the competition steps than using a value of magnitude equal to one for all the samples. However winner update will be different in both cases. That is because in the first case the reference vector of each unit is the weighted mean of samples in its Voronoi Set, while using magnitude equal to one, the final codewords are the simple mean of those samples.

Figure 3.6 shows the results of training a MSCL with the same dataset used in the Voronoi subsection (and the same number of units), using the following values for $\gamma = (0, 0.2, 1 \text{ and } 5)$. Magnitude function used was the absolute value of the first component of each data sample. This will force units towards rightmost Gaussian distribution of the dataset.

It can be noticed that $\gamma = 0$ distributes units nearby the density distribution. That is not exactly the density distribution because γ depends on the fraction of accumulated magnitude at each unit instead of its accumulated frequency. With $\gamma = 0.2$ some of the units moves towards zones with higher magnitude, but their distribution only follow the magnitude distribution when $\gamma = 1$ (this is the normal case). Value of $\gamma = 5$ concentrates units more densely than the normal case in zones with high magnitude, so that units have almost disappeared from the lower left Gaussian (the one with magnitude near zero).

This parameter, has similar effect in different data distributions and is useful in situations where magnitude variation is small from zones with high magnitude to zones with low magnitude.

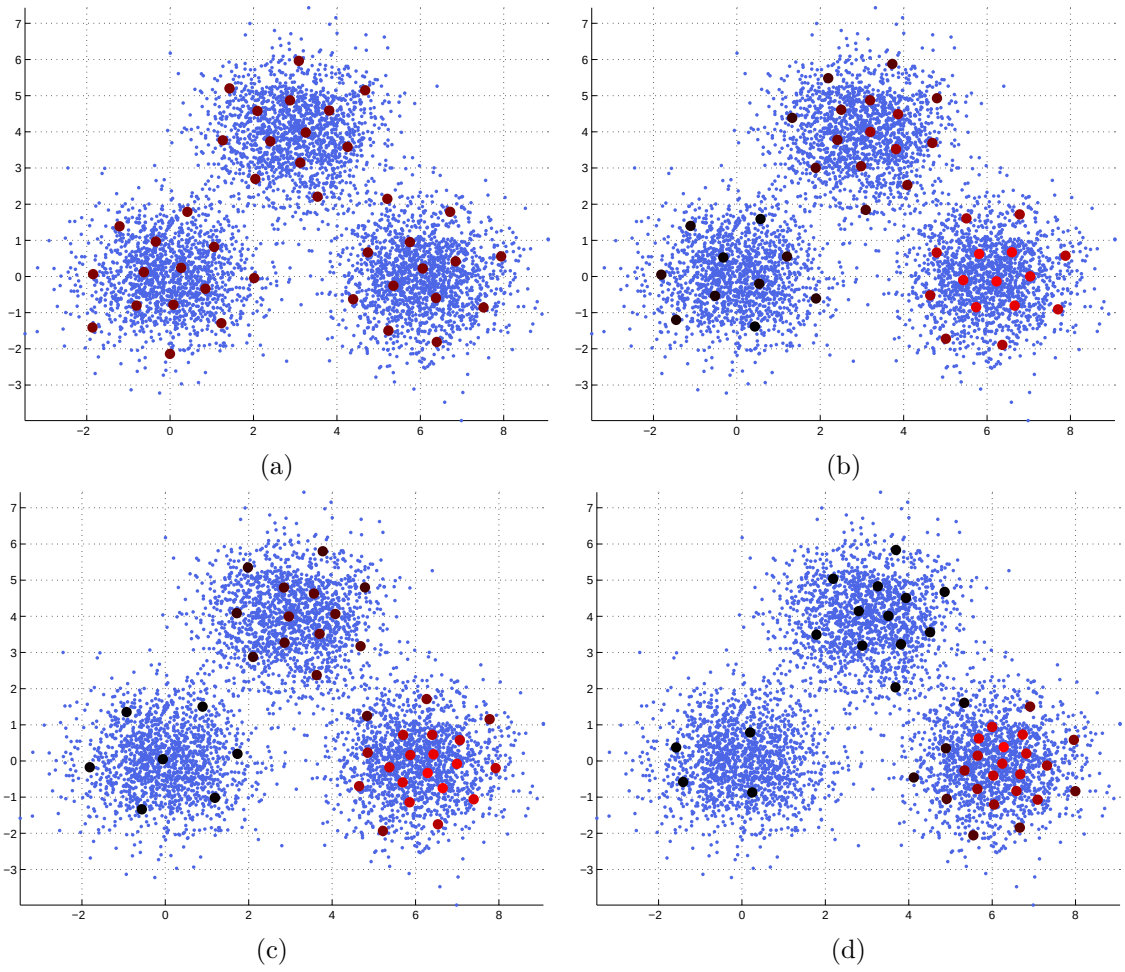


Figure 3.6: Effect of the value of γ during training a MSCL. Higher values enhance the importance of the magnitude during competition. In the figure we represented the following values for it: $(a)\ \gamma = 0$, $(b)\ \gamma = 0.2$, $(c)\ \gamma = 1$ (normal situation) and $(d)\ \gamma = 5$. Units are colored according the value of $mu_m(t)^\gamma$ (black means near zero values, and red the highest one).

3.3.7 Effect of the number of winners

The number of winners in the first competition by distance (K) was established to be equal to 2 in the rest of the chapters. This selection is based in multiple simulations executed with different values of K, analysing the final measures of the networks after training process. The problem chosen was the synthetic-gaussian distributions using data samples with 2, 10 and 30 dimensions. Results did not vary for dimensionality, so figure 3.7 only shows results for the problem with 10 dimensions. Influence in the number of units of the network was also explored, using 25, 50 and 100 neurons, but again the same behaviour was observed, so figure 3.7 only shows results for networks with 50 units.

The quality measures calculated in the networks were: WMSE, Entropy and DB-index, that are shown by columns in that figure. The different magnitude functions explored were three: first row of graphs shows results for MSCL with magnitude equal to the absolute value of x_2, second row of graphs show MSCL emulating FSCL, and the third row shows results for MSCL with Q-error. Horizontal coordinate is the value of K used in each experiment (averaged 10 times).

It is clear that K=2 shows the most reasonable behaviour, and if K is too high (K=50), the behaviour becomes unstable for the first magnitude function. With the other two magnitude functions it is not so clear, but as processing time for the competition step is larger as K increases, best selection is K=2.

3.4 Application examples

In order to demonstrate the effectiveness of the MSCL algorithm in oriented tasks, we perform several magnitude-function experiments in two types of data processing: Gaussian data quantization, and series interpolation. Five representative competitive learning algorithms are compared with MSCL along the experiments: frequency sensitive competitive learning (FSCL), Fuzzy c-means clustering (FCM), Neural Gas (NG), K-Means and Self-Organizing Maps (SOM). We do not pretend with these experiments to demonstrate that MSCL is the best strategy for the proposed application fields. The goal of the experiments is to show the focused behaviour of the MSCL with the different function magnitudes compared with the density focused behaviours of the other VQ methods.

3.4.1 Modelling Gaussian distributions

In this example a synthetic data set consisting of $P = 5000$ samples in a 2D plane ($\mathbf{x}(t) \in \mathbb{R}^2$) was generated from a mixture of three Gaussian distributions with means [0,0], [3,4]

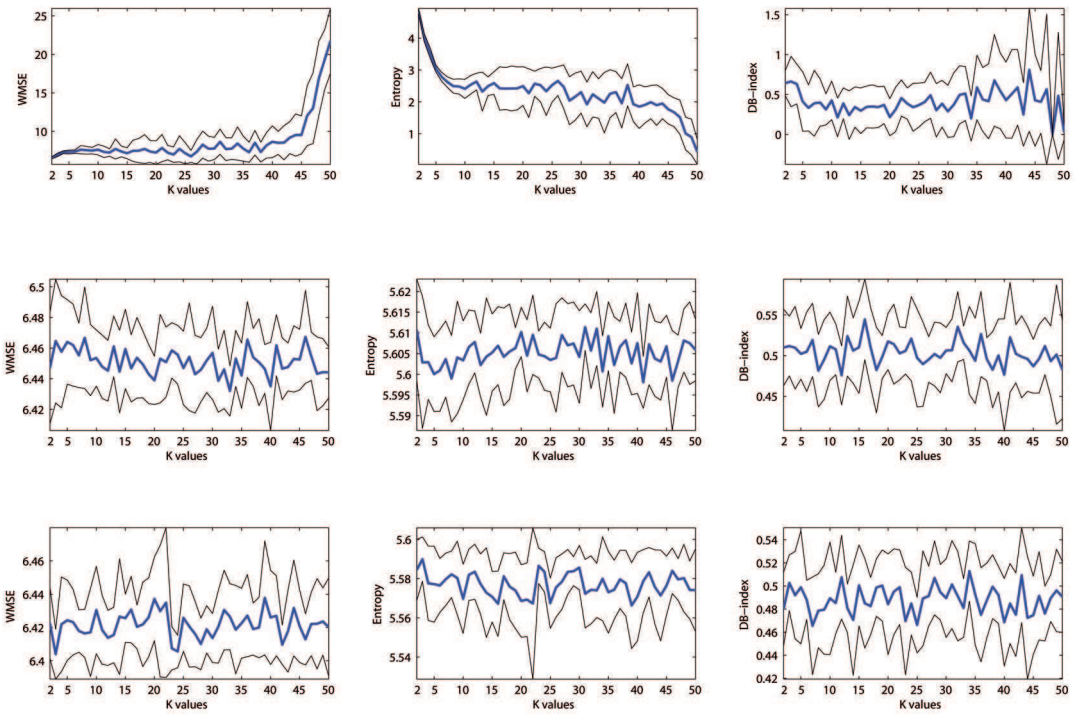


Figure 3.7: Analysis of the MSCL behavior for different K values in networks with 50 units. Three magnitude functions were explored: first row of graphs shows results for MSCL with magnitude in equation 3.26 (absolute value of x_2), second row of graphs show MSCL emulating FSCL, and the third row shows results for MSCL with Q-error. Graphs show the averaged final values (wide blue line) of the quality measures (WMSE, Entropy and DB-index) and the standard deviation values (narrow blue lines). The networks were simulated 10 times for averaging and were trained along 10 cycles with the synthetic-Gaussian problem. Horizontal coordinate is the value of K used in each experiment. It is clear that K=2 shows the most reasonable behaviour, and if K is too high (K=50), the behavior becomes unstable.

and [6,0], and covariance matrix [0.1 0; 0 0.1] for all of them. The fraction of samples placed in each cluster is 0.33 for the first and second distributions and 0.34 for the third. This data will be quantized by three different tests with 40, 80 and 160 prototypes ($M = \{40, 80, 160\}$).

For FCM algorithm, we used 2 as exponent for the membership partition matrix. Neural gas parameters were: initial step size of 0.5 and initial decay constant equal to N/2. K-Means were trained using a batch method. Finally SOM was a two-dimensional map initialized linearly with default parameters provided by SOM Toolbox [79]. We used 10

training cycles for all methods. To emulate FSCL, we set MSCL with magnitude function for each unit as the normalized winning-frequency of the unit, using initial and final learning rates of $\alpha_{ini} = 0.9$, and $\alpha_{final} = 0.1$. For MSCL, we used five predefined magnitude functions in different oriented tasks, all of them with values of *beta* so initial and final learning rates takes the following values: $\alpha_{ini} = 0.9$, and $\alpha_{final} = 0.1$.

Q-error magnitude function

The first magnitude function for MSCL is the mean Q-error in their Voronoi regions:

$$MF(m)(t) = mean(Q_{err}(t, m)) \tag{3.36}$$

As calculation of the magnitude function for each sample is computationally hard, it is only recalculated after each training cycle. The $Q_{err}(t, m)$ is the quantization error of the data samples belonging to the Voronoi region of unit u_m. This function tends to distribute prototypes over the data distributions to generate the same mean Q-error in all the units. This magnitude function will be used to generate a density estimation of the data distributions that is needed for the next magnitude functions.

Data density estimation

The next four magnitude functions are defined to focus the prototypes on the regions of the data distributions with dense or sparse presence of samples. For the former, we expect the prototypes to be concentrated in the means of the distributions. For the later, we expect prototypes to be distributed mainly far from the means, surrounding the distributions. For this purpose we need an estimation for the density of samples. In order to do this, an initial codebook (that we call density codebook) is trained to generate the estimation of the data density. The density estimation with $M = 80$ density codewords is shown in figure 3.8. It was obtained for MSCL with mean Q-error as magnitude function.

The reason of using this MSCL as the generator of the density codebook is based in the next reasoning. Lets suppose that the Voronoi area of each density codeword is similar, so we can approximate the density function as the number of samples of each codeword in its Voronoi region normalized by the total number of samples. This estimation of data density for these codewords provides a probability distribution that can be separated into two groups (Otsu method [58] was used to obtain a suitable threshold): codewords with higher density are marked with black points in figure 3.8, while those units without black points represent the codewords with density below the threshold. This resulting density

codebook separated in two groups, is fixed for all the simulations. It will be used as a reference table to define low density regions ($\mathcal{M}_{sparse} \in \mathcal{M}$), and high density regions ($\mathcal{M}_{dense} \in \mathcal{M}$) to calculate the four magnitude functions described ahead.

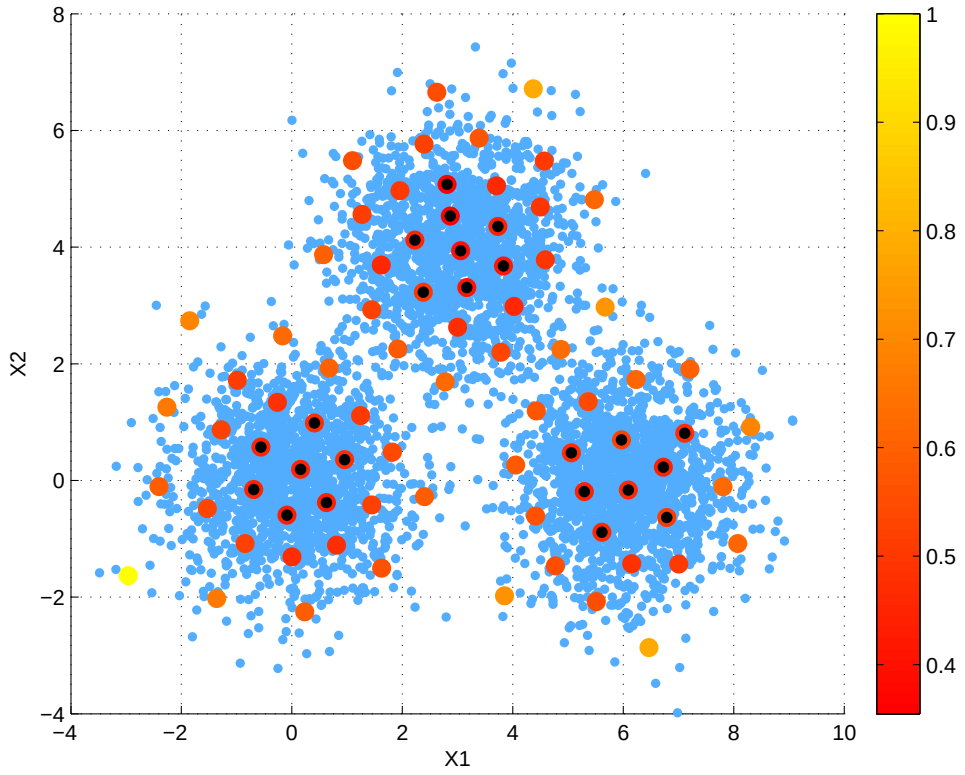


Figure 3.8: Example of representation with 80 units (trained with MSCL for mean Q-error as magnitude function) used to estimate the data density. The centred black points indicate the units corresponding to the dense-zone codewords (ω_{dense}), while the rest unit prototypes are assigned to the sparse-zone codewords (ω_{sparse}). The color bar represents the normalized-density (magnitude) values.

Contract and expansion magnitude functions

In order to force units to concentrate in these dense or sparse zones, we can apply two magnitude functions, that we call Contract1 and Contract2, to move prototypes to the dense zones, and two magnitude functions, called Expansion1 and Expansion2, to move prototypes to sparse zones. Many other definitions of magnitude functions are possible to focus prototypes in these zones, but we selected these four magnitude functions in a simple

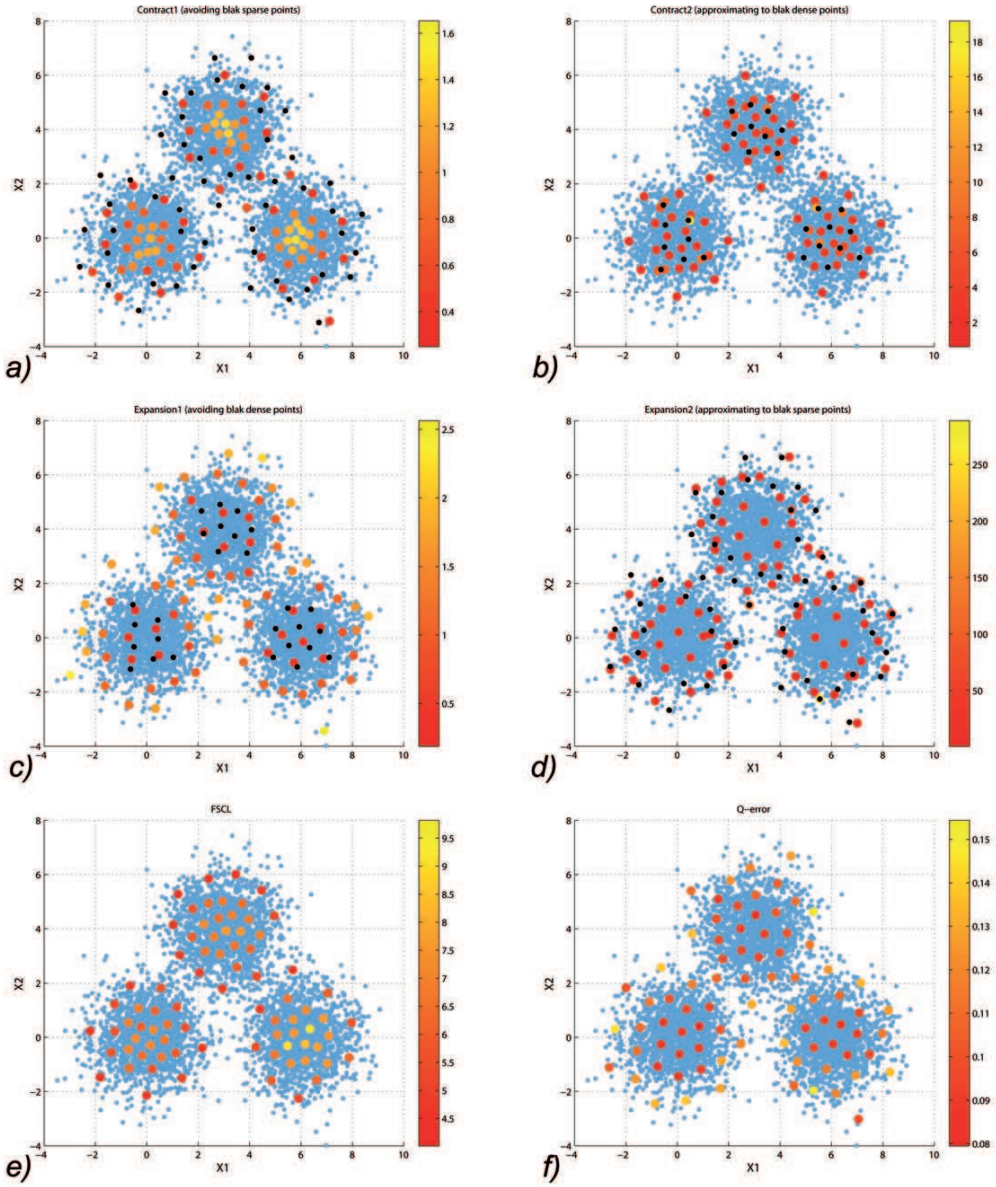


Figure 3.9: Gaussian example. Resulting representations of MSCL for contraction magnitude functions are shown in figures ***a*** and ***b***, MSCL with expansion magnitude functions in ***c*** and ***d*** MSCL with Q-err in ***f*** and FSCL in figure ***e***. The color bar represents the magnitude values assigned to the units. Black points represent the initial codebook for density estimation of the data, separated by Otsu method into dense (shown in figures ***b*** and ***c***) and sparse (shown in figures ***a*** and ***d***) sub-representations. Contract1 and Expansion1 (***a*** and ***c***) present magnitude functions that force units to avoid the corresponding black points. Contract2 and Expansion2 (***b*** and ***d***) present magnitude functions that force units to approximate to the black points.

form, as it was not our objective to determine which is the best selection of magnitude function for each task. Figure 3.9 shows an example of simulation results corresponding to the MSCL algorithm for different contraction and expansion magnitudes.

In MSCL example for Contract2, the magnitude function is chosen to concentrate units in data zones with high density, identified by ω_{dense}. First, for each unit u_m it is calculated its minimal distance (MD) to the zone of the dense-prototypes, $\mathcal{M}_{dense}$:

$$MD_m(t) = \min_{\substack{u_m \in \mathcal{M} \\ u_k \in \mathcal{M}_{dense}}} (\|\omega_k(t) - \mathbf{w}_m(t)\|) \tag{3.37}$$

And the magnitude function of each unit uses this distance normalized by the maximum for all units:

$$MF(m)(t) = 1 - \frac{MD_m(t)}{\max_{u_m \in \mathcal{M}}(MD_m(t))} \tag{3.38}$$

This magnitude function has maximum value 1 when the unit prototype coincides with one of the dense-prototypes, so units are impelled to compete for placing their prototypes over the black dense-representation, as can be appreciated in figure 3.9(b).

We can also force the contraction of the prototypes in dense zones by avoiding the sparse-prototype representation. The magnitude function in this Contract1 approximation is calculated:

$$MD_m(t) = \min_{\substack{u_m \in \mathcal{M} \\ u_k \in \mathcal{M}_{sparse}}} (\|\omega_k(t) - \mathbf{w}_m(t)\|) \tag{3.39}$$

$$MF(m)(t) = \frac{MD_m(t)}{\max_{u_m \in \mathcal{M}}(MD_m(t))} \tag{3.40}$$

This magnitude function has maximum value of 1 in one unit such as its distance to the $\mathcal{M}_{sparse}$ set is the maximum from all prototypes, so units are impelled to compete for placing their prototypes far from the black sparse-prototypes in figure 3.9(a). It is worth noting that these prototypes mainly tend to concentrate on the means of data distributions, while external distribution zones, marked with the black sparse-prototypes, are less represented.

For the expansion applications we aim to concentrate units in data areas with low data-density. The same magnitude functions used in contraction cases can be used by interchanging dense-prototypes by sparse-prototypes. The Expansion1 case shown in figure

3.9(c) is obtained by avoiding the dense-prototypes, while Expansion2 shown in figure 3.9(d) is obtained by approximation to the sparse-prototypes. These expansion cases tend to represent the boundaries of the clusters with more detail. Novelty detection applications can benefit from this behavior, as they precisely need to distinguish these boundaries with more detail, in order to identify if a new data sample belongs to the known data distribution or not.

For all methods mentioned before, three tests with 40, 80 and 160 prototypes ($M = [40, 80, 160]$) for initial random weights were trained with the dataset during 50 cycles for each algorithm. Along the training process three evaluation measures are calculated to represent their evolution and final values. The three tests were replicated 30 times and their evaluation measures averaged to avoid distortion due to the random initialization.

In order to evaluate the performance of the methods in generating a vector quantization task, and considering that data is not labelled in any form, we propose three unsupervised measures for evaluation: Davies-Bouldin Index (DB-index) [20], Shannon's informational entropy (normalized by the maximum value *log2(number of codewords)*) and the Weighted Mean Squared Error (WMSE).

Figure 3.10 shows the training evolution of the mean DB-index in the different methods (averaged in 30 simulations). The left graph shows the evolution for methods: MSCL with Q-error as magnitude, FSCL, FCM, NG, SOM and K-means. The right graph also shows these two methods as comparison, and the results for the different MSCL in contract-expansion examples. The minimum final values correspond to MSCL with Expansion1 and Q-error as magnitude functions. As expansion examples do not pretend to achieve an optimum VQ of the dataset, their DB-index evolution present final values higher than the other magnitudes. It is interesting to note that DB-indexes of Contract1 and Expansion1 examples that avoid the sparse-prototypes and dense-prototypes, respectively, present lower values than Contract2 and Expansion2 examples that approximate the dense-prototypes and sparse-prototypes respectively. This difference stems surely from the fact that the approximating strategy is less flexible in the prototype distribution (focusing in the fixed black or sparse prototypes) than the avoiding strategy.

In the top-left graph of figure 3.10, the MSCL with Q-error generates the lower DB-index, and surprisingly the Expansion1 magnitude function in the top-right graph can generate the minimum value. This magnitude function forces units to avoid the dense codewords in the center of distributions and generates a more detailed representation of the external parts of data distributions. MSCL, with mean Q-error as magnitude function, tends to give the best clustering representations measured by DB-index, that is not

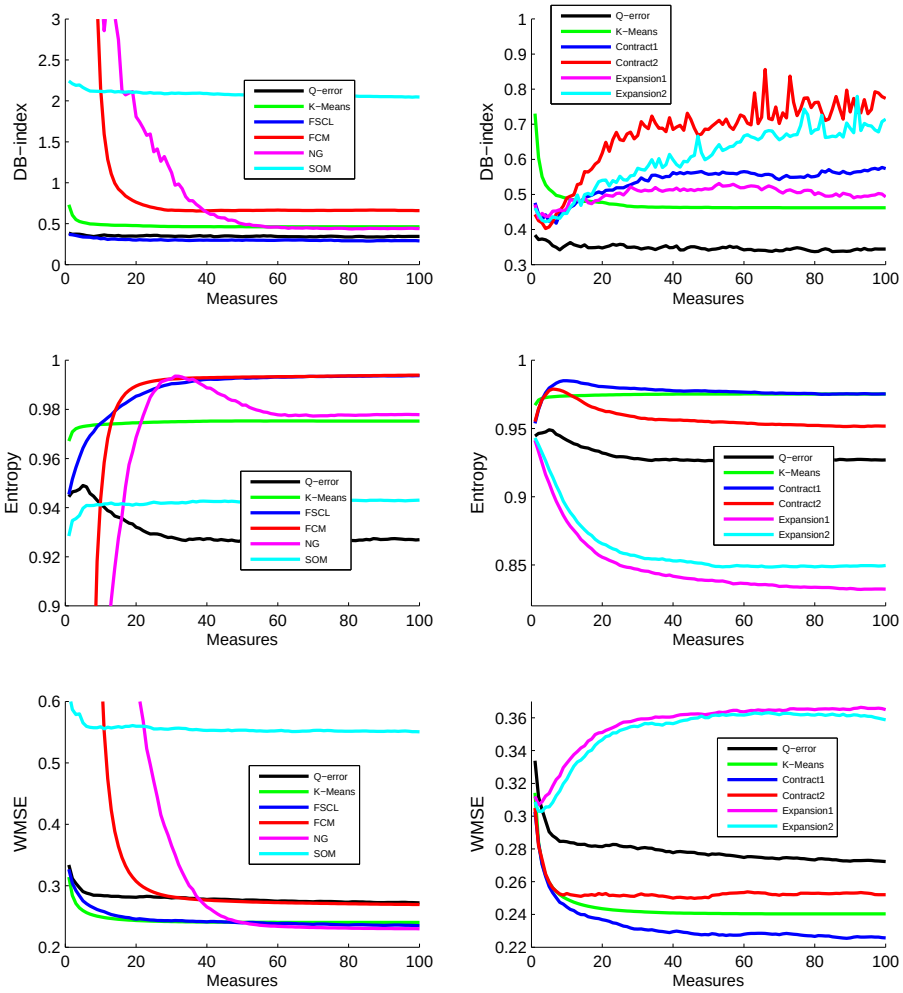


Figure 3.10: Evolution along training process during 50 cycles of the averages in 30 simulations of the DB-Index, Normalized Entropy and Weighted Mean Squared Error. The left column shows the evolution for methods: MSCL with Q-error, FSCL, FCM, NG, SOM and K-means. The right column shows also MSCL with Q-error and K-means as comparison, with the results for the different MSCL in contract-expansion examples.

surprising as DB-index is based on evaluating the homogeneous partition of mean Q-error among prototypes, so MSCL using Q-error as magnitude, which tends to accomplish this desired behavior, becomes the best in this index evaluation.

For normalized entropy in graphs of the second row, the best method is FSCL, as it

Method	BD Mean/Std	H Mean/Std	WMSE Mean/Std
Q-error	**0.385** 0.070	0.9157 0.0034	0.2743 0.0023
FSCL	0.491 0.188	**0.9972** 0.0003	0.2247 0.0010
Contract1	0.638 0.382	0.9739 0.0026	**0.2186** 0.0093
Contract2	0.837 0.425	0.9493 0.0077	0.2497 0.0116
Expansion1	0.492 0.147	0.8157 0.0094	0.3936 0.0124
Expansion2	1.065 0.699	0.8482 0.0080	0.3653 0.0140
FCM	0.673 0.286	0.9934 0.0016	0.2735 0.0052
NG	0.385 0.108	0.9776 0.0013	0.2307 0.0012
K-Means	0.473 0.130	0.9751 0.0027	0.2388 0.0041
SOM	2.093 0.108	0.9423 0.0036	0.5530 0.0124

Table 3.2: Mean and standard deviation, for 30 tests, of final measures: DB-index, Normalized entropy and WMSE, after training 40 units for the VQ task along 50 cycles. The codebook for the estimation of densities has 80 codewords and is shown in figure 3.8.

was expected. The other standard VQ methods tend to rise their entropies as they are methods focused in data density replication, but MSCL methods do not increase entropies, as their magnitudes are other than data density replication. The bottom graphs represent the Weighted MSE. In this case, the magnitude used for weighting the MSE of each unit of all methods is obtained as the equation 3.38, so all methods who tend to concentrate units in the center of the data distributions will present a decreasing evolution for their WMSEs (as it can be appreciated in the lower-left graph), being the MSCL methods with contract magnitude functions whose present the lower values of all (lower-right graph). However, the MSCL with expansion magnitude functions (Expansion1 and Expansion2) tend to push their units out of the centers of data distributions and the calculated WMSE tends to grow instead of decreasing (lower-right graph).

Tables 3.2, 3.3 and 3.4 show the mean and standard deviation (30 simulations) for the final values obtained in the three evaluation measures with the different algorithms using 40, 80 and 160 units. For the entropic measure, FSCL is always the best, as was expected by [1]. For the Weighted MSE, as this measure is obtained by applying normalized magnitudes obtained with equation 3.38, the MSCL methods with contracting magnitude functions (Contract1 and Contract2) tend to give the lower values in WMSE. For DB-index measure it is not so clear which method is the best in obtaining the lower value, besides the standard deviation values are quite large compared with those of entropy and WMSE. However, MSCL with Q-error presents in DB-index consistent lower values and standar

Method	BD Mean/Std		H Mean/Std		WMSE Mean/Std	
Q-error	0.3559	0.0639	0.9160	0.0021	0.1404	0.0012
FSCL	0.3842	0.1805	**0.9967**	0.0003	0.1131	0.0009
Contract1	0.5717	0.4761	0.9738	0.0024	0.1093	0.0053
Contract2	0.7375	0.4298	0.9640	0.0037	**0.1041**	0.0033
Expansion1	**0.3322**	0.1503	0.8201	0.0070	0.1962	0.0050
Expansion2	0.7019	0.3268	0.8568	0.0064	0.1794	0.0068
FCM	0.7874	0.4513	0.9910	0.0014	0.1584	0.0035
NG	0.3904	0.1462	0.9768	0.0013	0.1189	0.0007
K-Means	0.4208	0.1743	0.9760	0.0025	0.1233	0.0026
SOM	3.1550	0.1980	0.9451	0.0021	0.4285	0.0073

Table 3.3: Mean and standard deviation, for 30 tests, of final measures: DB-index, Normalized entropy and WMSE, after training 80 units for the VQ task along 50 cycles. The codebook for the estimation of densities has 80 codewords and is shown in figure 3.8.

Method	BD Mean/Std		H Mean/Std		WMSE Mean/Std	
Q-error	0.3429	0.1628	0.9337	0.0016	0.0641	0.0004
FSCL	**0.3179**	0.1570	**0.9960**	0.0003	0.0544	0.0006
Contract1	0.4487	0.3566	0.9636	0.0022	0.0518	0.0016
Contract2	0.5652	0.3807	0.9622	0.0017	**0.0497**	0.0009
Expansion1	0.4061	0.1473	0.8373	0.0032	0.0878	0.0017
Expansion2	0.5723	0.4350	0.8794	0.0024	0.0794	0.0057
FCM	0.8428	0.6305	0.9782	0.0018	0.1107	0.0030
NG	0.3443	0.1274	0.9782	0.0012	0.0580	0.0005
K-Means	0.3911	0.1562	0.9830	0.0024	0.0621	0.0014
SOM	3.6653	0.1224	0.9605	0.0015	0.2508	0.0059

Table 3.4: Mean and standard deviation, for 30 tests, of final measures: DB-index, Normalized entropy and WMSE, after training 160 units for the VQ task along 50 cycles. The codebook for the estimation of densities has 160 codewords and is shown in figure 3.8.

deviations, as it was expected cause the DB-index is based in Q-error estimation.

3.4.2 Interpolation application

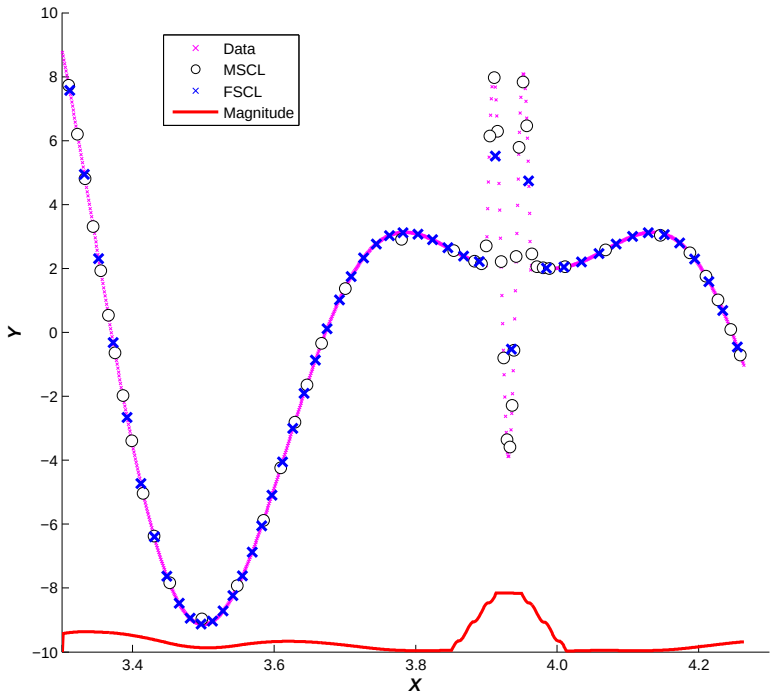


Figure 3.11: Interpolation example showing the FSCL and MSCL representations of the data series. The red line represents the magnitude value along the data series. FSCL does not represent the data in the high frequency perturbation as well as MSCL.

Next example consists in interpolating a designed data series to show that MSCL can be focused to any characteristic of the data. In this case, we focus the magnitude function to have a detailed representation of high frequency variations that can be found in the data series. Data set was generated by uniformly sampling a function with three piecewise function with a high frequency perturbation:

$$
\begin{aligned}
&LF(x) = -10\cos(12x) + 12\sin(10x),\\
&HF(x) = 2.1 + 6\sin(0.15x),\\
&y(x) = \begin{cases} LF(x), & \forall\ x \in \{(3.3, 3.9], [3.96, 4.3)\}\\ HF(x), & otherwise \end{cases}
\end{aligned}
$$

with $x = \{3.3, \ldots, 4.3\}$.

Two networks with 50 units (with a value of β so $\alpha_{ini} = 0.5, \alpha_{final} = 0.01$) were trained for 100 cycles with input data vectors (x, y), one with FSCL and the other with MSCL. The selected magnitude function is focused on detecting high frequency peaks. To accomplish it, $MF(m)$ is chosen as the average of the smooth function $SM()$ for data samples assigned to unit u_m at time t:

$$\begin{aligned} SM(x_k) &= smooth(|y(x_k) - y(x_{k-1})|) \\ MF(m) &= mean(SM(x_k)) \quad \forall x_k \in \mathcal{R}_m(t) \end{aligned} \tag{3.41}$$

The smooth function generates a 100 point moving average filter of the series of differences of y in two consecutive points.

Figure 3.11 shows that MSCL (black circles) is more efficient than FSCL (blue crosses) to represent high frequency peaks. The value of the magnitude function is also represented (red line) showing high values when $y(x)$ presents abrupt changes, and low values when $y(x)$ is almost constant.

Chapter 4

MS-SOM algorithm

4.1 Introduction

Soft competitive learning comprises a set of methods were more than a single neuron adapt on presentation of a sample pattern. These algorithms possess some features that are advantageous over hard competitive learning methods: avoiding unused ('dead') units, accelerating the learning phase, filling empty areas in the dataset space or avoiding local minima. Self Organizing Maps (SOM) [38] is one of these algorithms with the property of generating topographic organization of neurons in a grid of reduced dimension. This makes SOM useful for visualizing low-dimensional views of high-dimensional data, akin to multidimensional scaling. SOM has also been used for data classification (i.e. [51], [7]).

On the other hand, Magnitude Sensitive Competitive Learning (MSCL) [61] is a hard competing algorithm which has the capability of distributing the unit centroids following any user defined magnitude that may have no kind of relation with the data density (as it has been demonstrated in the previous chapter).

Comparing both algorithms, the main disadvantage of SOM neural networks against MSCL is that SOM only can distribute unit in direct function of the data density. Only magnification control methods ([81], [54]) present an alternative to SOM that allows the modification of the relation between data and weight vector density for a given model. However, it is important to highlight that in this kind of methods, final unit distribution is always somehow related with data density.

In this chapter we describe a new algorithm, *Magnitude Sensitive Self Organizing Map (MS-SOM)*, an hybrid between MSCL and SOM, which synthesizes the advantages of both methods. It preserves the topological properties of the input space and additionally,

distributes units following a target magnitude.

The remainder of this chapter is organized as follows. Section 4.2 describes the proposed MS-SOM algorithm. In section 4.3 the new algorithm is applied to three examples: a toy example with Gaussians to show the algorithm capabilities, 3D surface modelling, and data classification.

4.2 Magnitude Sensitive Self Organizing Maps

4.2.1 The algorithm

Magnitude definitions:

As in the MSCL algorithm, the user-defined magnitude function, $MF()$, acts as an extra information for the network, forcing neurons to represent with more detail those zones of data space with higher magnitude values. There exists mainly two situations depending on the data dependency of this function $MF()$: when magnitude is determined exclusively from input data, $MF(\mathcal{X})$, we define a magnitude vector, $\mathbf{mx}$, that is included as an extra input for the neurons of the map, however, when magnitude function also depends on neuron data, $MF(\mathbf{w}_m(t), \mathcal{X})$, we define for each neuron m an internal variable, $mu_m(t)$. These unit variables can be represented as a magnitude map $\mathbf{Mu}(t)$ with the same dimensionality of the map grid. Only in the second situation, the magnitude value of the winning neuron must be feed back to the rest of the neurons for their updating phase. The examples studied in next sections show both situations: 3D-surface example presents magnitude vector associated with input data, while Gaussian and classification examples present magnitude maps associated with neurons.

1. Initialization

Initial codebook $\mathcal{M}$ is formed by M weight vectors $\mathbf{w}_m$ $(m = 1 \cdots M)$ initialized linearly, forming a low dimensional grid (usually 2D). For the case when magnitude depends on neurons, we need to initialize the magnitude map in $t = 0$ with initial values for $\mathbf{Mu}(0)$. Their accumulated magnitudes are $macc_m$. Their initial values at $t = 0$ are:

$$mu_k(0) = MF(\mathbf{w}_m(0), m, \mathcal{X}) \tag{4.1}$$

$$macc_k(0) = mu_k(0) \tag{4.2}$$

2. Random selection of data samples

A sample data $\mathbf{x}(t) = (x_1, .., x_D)(t) \in \mathbb{R}^D$ is randomly selected at time t from the dataset $\mathcal{X}$ with P patterns.

3. Global unit competition

The unit i with minimum distance from its weights to the input data vector is selected as global winner in this first step.

$$i = \underset{m \in \mathcal{M}}{\operatorname{argmin}}(\|\mathbf{x}(t) - \mathbf{w}_m(t)\|). \tag{4.3}$$

At this point, we form the local winner set $\mathcal{S}, (\mathcal{S} \subset \mathcal{M})$ with the M_{grid} units belonging to the neighbourhood in the grid, of unit i in the MS-SOM map as:

$$\mathcal{S} = \{s_1, s_2, ..., s_{M_{grid}}\} \tag{4.4}$$

For example, in a two dimensional grid with hexagonal representation, M_{grid} would have a value of 7, for the winner unit and its six closest neigbour units around.

4. Local unit competition

Winner unit j is selected from units belonging to $\mathcal{S}$, as the one that minimizes the product of its magnitude value with the distance of its weights to the input data vector, one of these equations:

$$j = \begin{cases} \underset{u_s \in \mathcal{S}}{\operatorname{argmin}}(mu_s(t)^\gamma \cdot \|\mathbf{x}(t) - \mathbf{w}_s(t)\|) \\ \underset{u_s \in \mathcal{S}}{\operatorname{argmin}}(macc_s(t)^\gamma \cdot \|\mathbf{x}(t) - \mathbf{w}_s(t)\|) \end{cases} \tag{4.5}$$

The use of mu in local competition is more adequate than $macc$ when the goal of training is Q_{err} reduction while $macc$ is better to reduce the entropy.

5. Winner and magnitude updating

For all units in the map, weights and magnitude are adjusted iteratively for each training sample, following $(m = 1 \dots M)$:

$$mf(t) = \begin{cases} mx(t), & \text{if used a magnitude vector (\textbf{mx}).} \\ mu_j(t), & \text{otherwise.} \end{cases} \tag{4.6}$$

$$macc_m(t+1) = macc_m(t) + mf(t) \cdot h_{mj}(t) \tag{4.7}$$

$$\alpha_m(t) = \left(\frac{mf(t) \cdot h_{mj}(t)}{macc_m(t+1)} \right)^{\beta} \tag{4.8}$$

$$\mathbf{w}_m(t+1) = \mathbf{w}_m(t) + \alpha_m(t) \left(\mathbf{x}(t) - \mathbf{w}_m(t) \right) \tag{4.9}$$

$$mu_m(t+1) = \begin{cases} mu_m(t) + \alpha_m(t) \left(mx(t) - mu_m(t) \right), & \text{if \textbf{mx} is used.} \\ MF(\mathbf{w}_m(t+1), m, \mathcal{X}), & \text{otherwise.} \end{cases} \tag{4.10}$$

In the above equations $h_{mj}(t)$ is the neighbourhood kernel around the winner unit j at time t. This kernel is a function depending on the distance of map units j and m in the map grid, and $mu_m(t)$ is the value of the magnitude at unit m. Finally, $\alpha_m(t)$ is the learning factor, γ defines the strength of the magnitude during the competition and β is a scalar value between 0 and 1. Observe in eq. 4.10 that if magnitude is presented as an extra input, the magnitude of the unit is updated as any other weight.

6. Stopping condition

Training is finished when a termination condition is reached. It may be the situation when all data samples has been presented to the MS-SOM neural network along certain number of cycles, T, or any other function that could measure the training stabilization.

4.2.2 Analysing of the algorithm

Competition:

Competition for the Best Matching Unit (BMU) includes a local competition step taking into account the magnitude, that forces units to move towards space regions of higher value of magnitude. Neurons with high values of magnitude are less competitive than those with low values (eq.4.5), so space zones with larger magnitude recruit more neurons in their representations.

Learning:

Learning factor $\alpha_m(t)$ for each unit depends on:

1. The value of the magnitude $mf(t)$ associated to each sample data. High magnitude produces high changes in unit weights, while values near zero produces practically no learning.

2. The distance from each unit to the winner unit. The importance of this factor is modulated by the kernel function h_{kj}. Higher distance means lower learning.

3. The accumulated magnitude at the unit. It is related to the firing history of each unit. High accumulated magnitude means high learning up to the moment, and therefore unit becomes practically static.

4. The value of β, the forgetting factor. Using the definition of learning factor of (4.9), when β is equal to one, units' weights become the running weighted mean of the value of the data samples belonging to its Voronoi region, and adjacent regions (weighted according to its neighbourhood). On the contrary, lower values of β means that recent patterns have higher importance in the running weighted mean. In the limit case ($\beta = 0$), each unit would become the last presented sample: $\mathbf{w}_m(t+1) = \mathbf{x}(t)$

Magnitude Map:

As MS-SOM generates a low dimensional grid structure, it is possible to draw a magnitude map in low dimension. This map has the same dimensions of the grid, with a value for each unit equal to its corresponding magnitude. Figures 4.1(a) and 4.2(c) show examples of magnitude maps. If magnitude is only data dependent, the weights of neurons in the magnitude input can be used to generate the magnitude grid. If magnitude depends on neuron, we can use the magnitude map **Mu**.

4.3 Application examples

4.3.1 Modelling Gaussian distributions

In this example we test the performance of a MS-SOM with four different types of magnitude functions, compared with a SOM.

We use a synthetic data set consisting of $N = 5000$ samples in a 2D plane ($\mathbf{x}(t) \in \mathbb{R}^2$) drawn from a mixture of three Gaussian distributions with means [0,0], [3,4] and [6,0], and covariance matrix [0.1 0; 0 0.1] for all of them. The fraction of samples placed in each cluster is N/3.

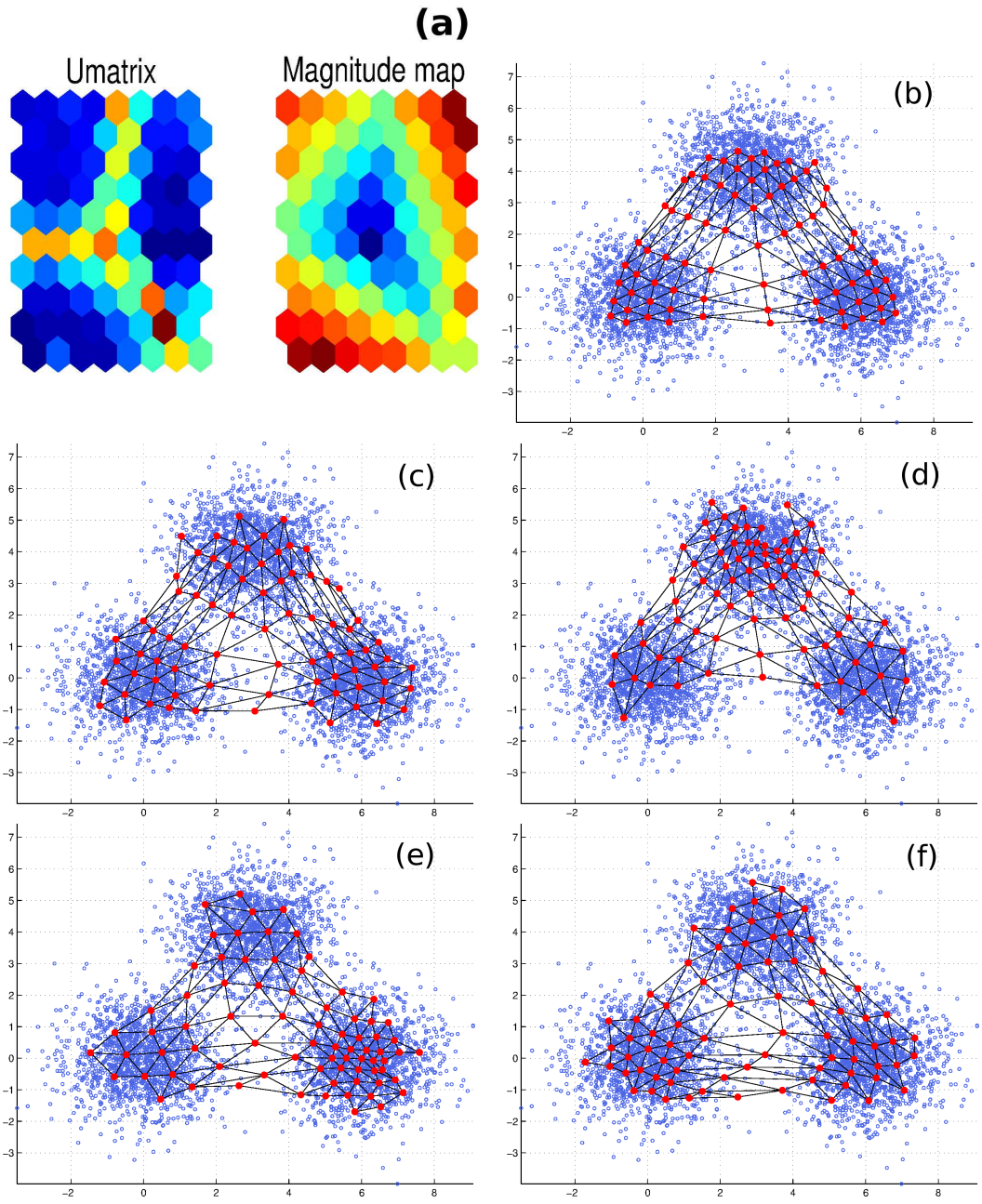


Figure 4.1: *Gaussian example.* (a) U-matrix and Magnitude map of MS-SOM using MF_4 as magnitude function. (b) Trained SOM. MS-SOM trained with MF_1 (c), with MF_2 (d), with MF_3 (e) and trained with MF_4 (d).

Units	*Algorithm*	*Constant*	X_2	$dist(0,0)$	$dist(X_{mean})$
40	*SOM*	0.574	0.726	0.688	0.692
	MSSOM	0.474	0.550	0.540	0.536
80	*SOM*	0.424	0.524	0.515	0.503
	MSSOM	0.357	0.435	0.402	0.404
160	*SOM*	0.296	0.361	0.358	0.353
	MSSOM	0.246	0.273	0.274	0.275

Table 4.1: Table shows the mean values in 100 tests of the *Weighted Mean Square Error (WMSE)* calculated in three codebooks (sizes 40, 80 and 160) after applying SOM and MS-SOM trained with four magnitude functions. WMSE is always lower in MS-SOM independently of the magnitude function used.

SOMs are trained using a Gaussian function for $h_{kj}(t)$ with neighbour ratios within [3,0.05] and a learning factor that decreases exponentially with time. Three SOMs are initialized linearly in the data space using codebooks of 40, 80 and 160 units.

MS-SOMs have the same number of units than SOM (also uses 40, 80 and 160), use the same initial codebooks, $h_{kj}(t)$ and a value of $\beta = 1$. We apply four different magnitude functions, that depend on unit weights:

1. Constant value: $MF_1(\mathbf{w}_m(t), m, \mathcal{X}) = 1$.
2. Distance to ordinate axis : $MF_2(\mathbf{w}_m(t), m, \mathcal{X}) = abs(w_{m,2})$.
3. Distance to point (0,0): $MF_3(\mathbf{w}_m(t), m, \mathcal{X}) = \| \mathbf{w}_m \|$.
4. Distance to the mean of dataset: $MF_4(\mathbf{w}_m(t), m, \mathcal{X}) = \| \mathbf{w}_m - \mathbf{x}_{mean} \|$.

Figure 4.1 shows some results for SOM and MS-SOM (with 80 units) of the grid representation over the data space. Figure 4.1(a) shows the corresponding U-matrix (Matrix of distances between neighbouring units), and the magnitude map for one MS-SOM. Figure 4.1(b) shows the typical result of a trained SOM were units tend to allocate their centroids in areas with higher data density.

MS-SOM neural network in Fig. 4.1(c) used a constant value for magnitude equal to one, so that magnitude function have no effect on final training. Units only compete by distance. As in the SOM case, units are centered in zones with high density. However its distribution is not so affected by the 'border effect', of the SOM representation. That is because the learning factor is different for each unit in MS-SOM and $\alpha_m(t)$ depends on

the activation frequency of unit m. In Figure 4.1(d) units avoid the ordinate axis, and in Figure 4.1(e) units avoid the(0,0) point.

Figure 4.1(f) shows a more expansive MS-SOM than using constant magnitude. Magnitude of units becomes higher as their centroids are farther from mean of dataset and units focus on these areas, although they have low data density.

We use Weighted Mean Squared Error (WMSE) as a measure of quantization quality. It is the weighted mean of the quantization squared error, where weight factors in the trained network are the values of mf (in eq.(4.6)), with $\mathcal{V}_m$ being the Voronoi set of the unit m:

$$WMSE(\mathcal{X};\mathcal{M}) = \frac{\sum_{\substack{m\in\mathcal{M}\\ \mathbf{x}\in\mathcal{V}_m}} mf \cdot \|\mathbf{x}-\mathbf{w}_m\|}{\sum_{\mathbf{x}\in\mathcal{X}} mf} \tag{4.11}$$

Table 4.1 shows the Weighted Mean Square Error (WMSE) calculated in three codebooks (sizes 40, 80 and 160) after applying SOM and MS-SOM trained with four magnitude functions indicated above. Results are different in the four magnitude functions because weights of each sample change depending on the selected function.

In all the cases MS-SOM surpass SOM, getting lower weighted quantization error. It is significant that in the case of the constant one magnitude function, MS-SOM is better than SOM, because 'border effect' is lower in MS-SOM.

4.3.2 Classification

Dataset in classification problems consists on P samples $\mathbf{x}(t) \in \mathbb{R}^D$ separated in K possible classes, $C \in \{C_1, C_2, \ldots, C_K\}$. Each sample has a label that indicates the class where the sample belongs to (see eq. 4.12) and is provided to the neural network during training, so it is able to provide class information for magnitude calculation at the units.

We will compare SOM and a MS-SOM that focus units in zones with high missclassification error. The process is as follows:

1. Vector data in the sample dataset are joined with the class label vector:

$$\mathbf{y}(t) = (x_1, \ldots, x_D, c_1, \ldots, c_K) \in \mathbb{R}^{(D+K)}, \tag{4.12}$$

being $c_k = 1$ if $\mathbf{x}(t) \in C_k$, or $c_k = 0$ if it belongs to other class.

2. Data samples are normalized for the first D components, the only components considered during the first competition step based in distance. Last part of vector $\mathbf{y}$ $(c_1, \ldots, c_K)$ is masked but it is updated during training.

3. SOM and MS-SOM are trained using $\mathbf{y}(t)$ as data inputs selected randomly. Magnitude function for MS-SOM depends on each unit and has the following value:

$$mu_m = K \cdot \frac{1 - max(\xi_m)}{(K-1)} \tag{4.13}$$

where ξ_m is the vector formed by the last K components of weight vector $\mathbf{w}_m$. This vector, ξ_m, acts as a counter of samples of each class captured by unit. By this way, magnitude is 0 if unit only have data samples of one class, or close to 1 in the situation of maximum confusion between the K classes. Then, $(max(\xi_m) = 1/K)$,

4. After training, the class assigned to each unit is:

$$class() = argmax(\xi) \tag{4.14}$$

In this classification comparative, we used three data sets: the Iris Dataset [31] and two downloaded from the Proben1 library [67]. First one consists of 150 samples from three species of Iris (Setosa, Virginica and Versicolor). The second dataset presents 6 types of glasses; defined in terms of their oxide content (i.e. Na, Fe, K, etc). The Third dataset is based on patient data to decide whether or not a Pima Indian individual is diabetes positive. Number of samples, inputs and classes are specified for each problem in Table 4.2.

SOM and MS-SOM were trained with the same parameters, with ratios within [3,0.05] using a Gaussian neighbouring function and a decreasing learning factor. Both neural networks received the same linear initialization. Map sizes for all the problems are displayed in column *Map* of Table 4.2. MS-SOM uses a value of $\beta = 1$.

Table 4.2 shows the mean classification error (CE) and the mean Weighted MSE (WMSE) averaged in 20 trainings with each dataset. CE is the total number of sam-

Problem	*Samples/Inputs*	*Classes*	*Map*	CE_S	CE_{MS}	$WMSE_S$	$WMSE_{MS}$
Iris	150 / 4	3	[5x3]	0.012	0.002	0.643	0.556
Glass	214 / 9	6	[6x4]	0.244	0.137	1.484	1.306
Diabetes	768 / 8	2	[10x8]	0.076	0.049	1.493	1.366

Table 4.2: Mean classification error (CE) and Weighted Mean Square Error (WMSE) for SOM (with sub-index S) and MS-SOM (MS) obtained after training both algorithms with the three datasets. Additionally number of samples, number of inputs, classes, and map size is displayed for each problem.

ples associated to an erroneous class after each test, divided by the number of samples in the dataset. Columns CE_S and CE_{MS} display classification errors for SOM and MS-SOM respectively. Columns $WMSE_S$ and $WMSE_{MS}$ are the equivalent for the Weighted Mean Square Error.

It is clear that in the three problems, MS-SOM with units focused in the limits between classes is able to distinguish more accurately the class to which each sample belongs to (it has lower CE error). The reason is that MS-SOM leave few units in areas with no class confusion (where classification error is null) while many of the units tend to be in the limit between classes. On the other hand, WMSE reflects the quantization error, focussing in areas of high magnitude. This measure is lower in MS-SOM algorithm, what means that its centroid density is higher in the decision regions, giving as result a better performance in the classification task.

Figure 4.2 shows a MS-SOM trained with the iris dataset: (a) Map with colours depending on the classes for each unit (interpolating colours mean that a unit has samples from different classes), (b) map with the final assigned class for each unit, and (c) magnitude associated to each unit. In the magnitude map, limits between the three classes are more clearly represented because MS-SOM tends to distribute units in the decision regions between contiguous classes.

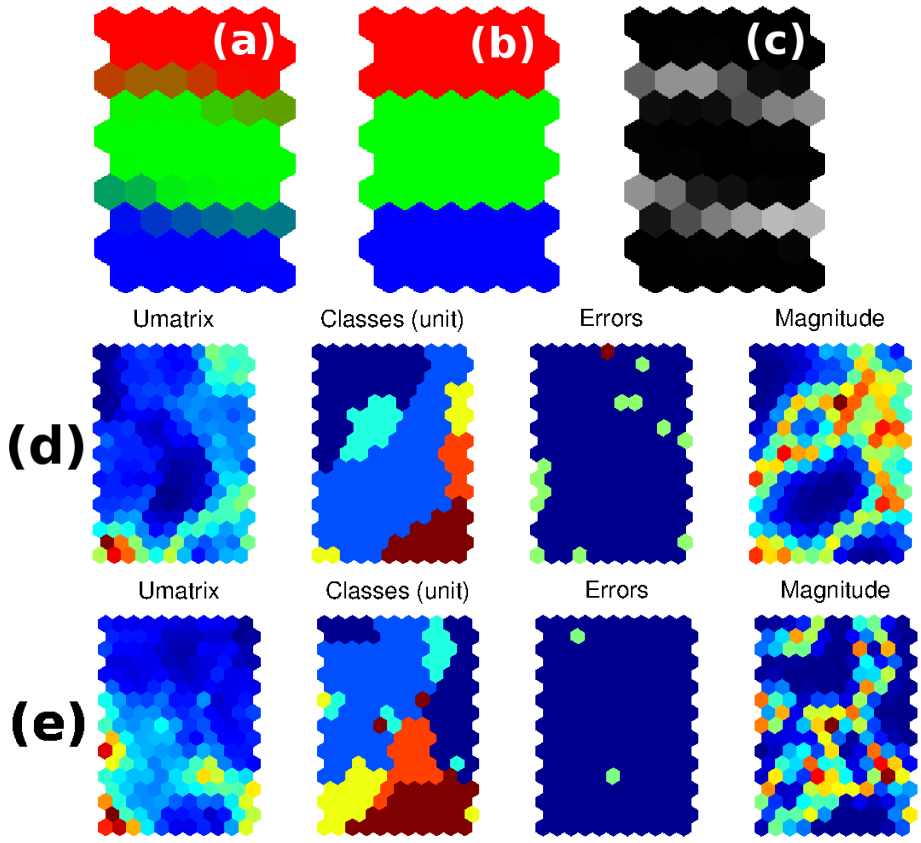


Figure 4.2: *Classification results for MS-SOM. Iris example:* (a)map with colours depending on the classes for each unit (interpolating colours mean that a unit has samples from different classes), (b) map with the final assigned class for each unit, and (c) magnitude associated to each unit (clearer grey means higher magnitude). In this representation, the map size (10x6 units) was bigger than the one used in the comparative to highlight the value of the magnitude in zones of high class confusion. *Glass example*: (d)Results of training a (17x11) grid with SOM. (e) Corresponding results of MS-SOM.

Chapter 5

Masked MSCL algorithm

5.1 Introduction

In chapter 3 we have explained the MSCL algorithm in detail. MSCL as many competitive learning algorithms use homogeneous dataset in the sense that each of the samples has to have the same dimension (the number of components), and the trained neural network also has the same dimension. Similar issue affects MS-SOM.

However sometimes, due to the nature of the dataset, some of the data samples may have unknown values for any of its components. This forces to some kind of preprocessing that usually introduces undesired artifacts during training. There may be different reasons for the inconsistency in the dataset's component size. Data collected in different periods of time or by different entities may be inconsistent. For instance meteorological data may lack of some variables as wind velocity when remounting may years ago, or just come from different meteorological stations. Then, it is no possible to process directly this dataset with common competitive learning methods. To do it, incomplete components in some of the samples must be dealt in some way. Another example is statistical information about citizens in different countries. Even though there are international entities that are doing a huge effort to unify measures, they are usually different in some way.

Here we present a masked version of MSCL that is able to deal with data samples of different size (we speak of 'masked' data). To use this algorithm we will consider that each data sample consists in two vectors, $\mathbf{x}(t) = (x_1, \ldots, x_D)(t) \in \mathbb{R}^D$ the data vector itself (with the maximal possible dimension of a data sample D), and its corresponding mask $\mathbf{msk}(t) = (msk_1, \ldots, msk_D)(t) \in \mathbb{R}^D$. The mask is a vector with ones in the valid components of $\mathbf{x}(t)$ and zeros for the rest.

The new algorithm will imply the use of vectors of length equal to D instead of scalars for the accumulated magnitude of each unit ($\mathbf{macc}_i(t) = (macc_{i1}, \ldots, macc_{in})(t) \in \mathbb{R}^D$), and also for the learning vector ($\mathbf{alpha} = (\alpha_1, \ldots, \alpha_n) \in \mathbb{R}^D$).

5.2 The masked MSCL algorithm

The next subsections describe the algorithm in an sequential mode, where units are adjusted after a presentation of each sample.

5.2.1 Initialization

Initialized the neural network $\mathcal{M}$ to contain M units:

$$\mathcal{M} = \{u_1, u_2, \ldots, u_M\} \tag{5.1}$$

with unit weights $\mathbf{w}_m \in \mathbb{R}^D$ (corresponding to unit u_m) the value of randomly selected inputs from the dataset:

$$\mathbf{w}_m(0) = \mathbf{x}(m) \quad m = 1 \ldots M \tag{5.2}$$

Then, unit magnitude might be initialized from the magnitude function depending only on unit parameters (equation 5.3a), or alternatively by the value of the magnitude at the selected sample data for each unit (eq. 5.3b). Obviously, this function has to be able to deal with different sized input vectors, and return a scalar value.

$$mu_m(0) = MF(\mathbf{msk}(t) \circ \mathbf{w}_m(0), < m >), \tag{5.3a}$$

$$mu_m(0) = mx(m), \tag{5.3b}$$

Finally, initial values for the magnitude accumulated vectors are:

$$\mathbf{macc}_m(0) = mu_m(0) \cdot \mathbf{msk}(m) \tag{5.4}$$

5.2.2 Random selection of data samples

A sample data $\mathbf{x}(t)$ and its corresponding mask, $\mathbf{msk}(t)$ is randomly selected at time t from the dataset. This process will be repeated until every data has been presented to the masked MSCL neural network.

5.2.3 Global unit competition

K units with minimum distance from their masked weights to the masked input data vector are selected as winners in this first step. In this case distance is calculated considering only valid components in $\mathbf{msk}(t)$ for both vectors. Once again we form the $\mathcal{S}$ set. $\mathcal{S} \in \mathcal{M}$ such that:

$$\|\mathbf{msk}(t) \circ (\mathbf{x}(t) - \mathbf{w}_s(t))\| < \|\mathbf{msk}(t) \circ (\mathbf{x}(t) - \mathbf{w}_m(t))\| \ \ \forall u_s \in \mathcal{S} \wedge u_m \notin \mathcal{S} \ . \tag{5.5}$$

5.2.4 Local unit competition

In the local competition step, winner unit j is selected from units belonging to $\mathcal{S}$ as the one that minimizes the product, component by component, of its magnitude value as in equation 5.6-top or the accumulated magnitude, eq. 5.6-bottom, using the distance of its weights to input data vector calculated in the p valid components. It follows:

$$j = \begin{cases} \underset{u_s \in \mathcal{S}}{\operatorname{argmin}} \|mu_m^{\gamma} \cdot (\mathbf{msk}(t) \circ (\mathbf{x}(t) - \mathbf{w}_m(t)))\|, & \text{or} \\ \underset{u_s \in \mathcal{S}}{\operatorname{argmin}} \|\mathbf{macc}_m(t)^{\gamma} \circ (\mathbf{x}(t) - \mathbf{w}_m(t))\| \end{cases} \tag{5.6}$$

The use of mu in local competition is more adequate than **macc** when the goal of training is Q_{err} reduction while **macc** is better to reduce the entropy.

5.2.5 Winner update

First it is necessary to calculate the value of the magnitude vector associated to the input sample $\mathbf{mx}(t)$. It may have two definitions depending if the magnitude function depends directly on the input, or it is calculated from the magnitude at the BMU(t) of the input sample:

$$\mathbf{mx}(t) = \begin{cases} MF(\mathbf{msk}(t) \circ \mathbf{x}(t)) \cdot \mathbf{msk}(t), & \text{if depends on sample magnitude} \\ mu_j(t) \cdot \mathbf{msk}(t), & \text{if depends on BMU's magnitude} \end{cases} \tag{5.7}$$

This vector is used to update the accumulated magnitude at winner unit:

$$\mathbf{macc}_j(t+1) = \mathbf{macc}_j(t) + \mathbf{mx}(t) \tag{5.8}$$

Then, only valid components (those with $msk_d = 1$) of winner weights are updated:

$$w_{jd}(t+1) = w_{jd}(t) + \alpha_d \left(x_d(t) - w_{jd}(t) \right), \quad d : msk_d = 1 \tag{5.9}$$

where **alpha** is the learning factor vector for the winner calculated as the element-wise division between **mx** and $\mathbf{macc}_j$ powered to β. Using this definition of $\mathbf{alpha}(t)$, only valid components at time t of winner weight are updated. The value of its k component is:

$$\alpha_k(t) = \begin{cases} \left(\frac{mx_k(t)}{macc_{jk}(t+1)} \right)^{\beta}, & \text{if } msk_k = 1, \\ 0, & \text{otherwise} \end{cases} \tag{5.10}$$

5.2.6 Magnitude update

Only winner's magnitude is adjusted iteratively for each training sample, following eq.3.23 if magnitude is given by a value associated to each input sample, or eq.3.24 otherwise. The mean value of **alpha** in the valid components is used as mu_j is an scalar:

$$mu_m(t+1) = \begin{cases} mu_m(t) + mean(\mathbf{alpha}(t)) \cdot \left(mx(t) - mu_m(t) \right), & \text{if } MF(\mathbf{x}(t)) \text{ is used.} \\ MF(\mathbf{w}_m(t+1), < m >), & \text{otherwise.} \end{cases} \tag{5.11}$$

5.2.7 Stopping condition

Training finish when a termination condition is reached (this condition may be the same as in the MSCL algorithm).

5.3 The masked MS-SOM algorithm

The masked MS-SOM algorithm follows the same steps and equations as the masked MSCL algorithm excepting in the following points:

- **mx** takes different values for different units taking into account the unit neighboring function, h_{ij} following:

$$\mathbf{mx}_i(t) = \begin{cases} h_{ij} \cdot MF(\mathbf{msk}(t) \circ \mathbf{x}(t)) \cdot \mathbf{msk}(t) \\ h_{ij} \cdot mu_j(t) \cdot \mathbf{msk}(t) \end{cases} \tag{5.12}$$

Unit neighboring function may be defined depending on the grid distance between each unit i and the winner j in different ways, for instance:

$$h_{ij} = exp\left(\frac{distance(i,j)^2}{2\sigma^2}\right) \tag{5.13}$$

- **alpha** and **macc** are calculated for every unit, not only for the winner.
- All units are updated at each step.

5.4 Experimental results

We perform a simple example to demonstrate the capability of the MSCL and MS-SOM in their 'masked' implementation to deal with incomplete data. To do it, we first set up a synthetic data set $\mathcal{X}$, consisting of $P = 3000$, 3D samples ($\mathbf{x}(t) \in \mathbb{R}^3$) drawn from a mixture of three Gaussian distributions with means [0,0,0], [1,1,1] and [0 0.35 0.65], and covariance matrix [0.02 0 0; 0 0 0.02; 0 0.02 0] for all of them. All of the three gaussians have 1000 samples.

Additionally, we generate a 'mask' matrix with the same size as $\mathcal{X}$, so that its i-th row vector corresponds to the mask of $\mathbf{x}(t)$, to indicate if that component is valid or not. Each component of this mask vector takes a value of 1 (with probability of 70%) or 0 (with probability equal to 30%).

Figure 5.1 shows the dataset used in the algorithms, and the corresponding mask.

We trained a MSCL and a MS-SOM neural networks in two cases, directly with dataset $\mathcal{X}$, an also using the 'mask' matrix. Codebooks neural networks have 10 units in both cases. For training, we tested two different magnitude functions. First a constant value equal to one (it is the same than training directly with the basic CL algorithm, and SOM). Tests were repeated using the absolute value of first component of each sample as the magnitude function.

Figure 5.2 shows final results for training the MSCL in the four cases (two magnitude functions, and direct(*graphs in top row*)/masked(*graphs in bottom row*) training). It can be seen that there is no appreciable difference between the use or not of the mask. Using magnitude equal to one, units spread homogeneously in the three Gaussian distributions in both cases. Using the magnitude funtion as $MF(t) = abs(x_1(t))$, units are mainly allocated in the Gaussian with higher value of first component of each sample.

Similar situation can be appreciated in Figure 5.3, that shows final results for training the MS-SOM in the four cases (two magnitude functions, and direct(*top*)/masked(*bottom*)

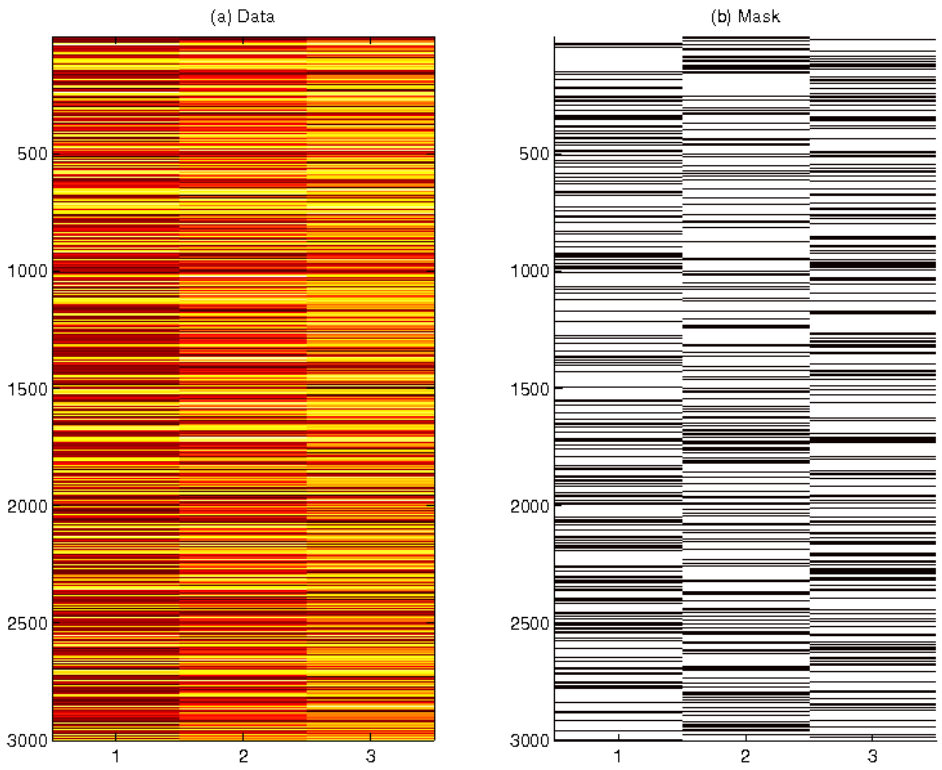


Figure 5.1: Dataset matrix (*left*) and corresponding mask(*right*). This dataset corresponds to the three 3D Gaussian distributions. Mask matrix indicates valid components of each sample (in white), or invalid (30% of components, in black).

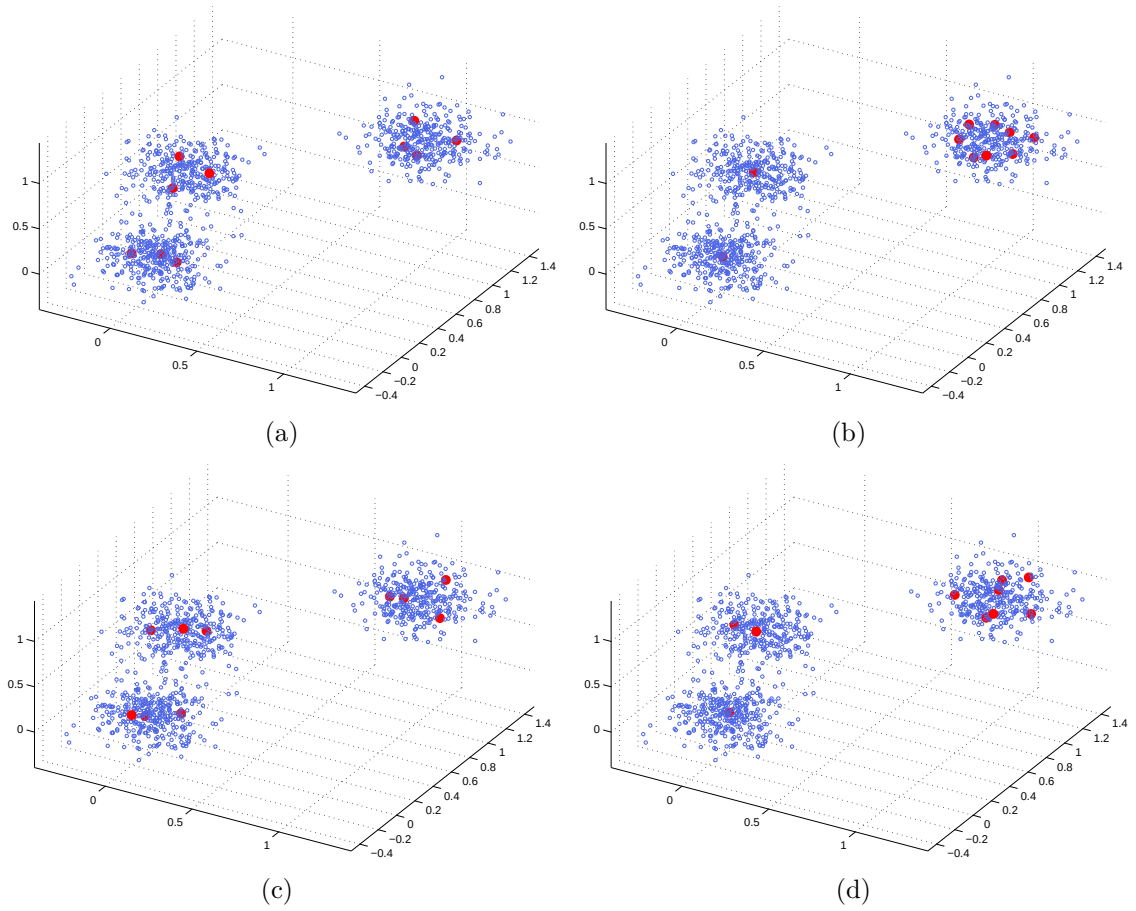


Figure 5.2: Final results of a trained MSCL with 10 units using two magnitude functions. On top, row figures represent direct training without taking the mask into consideration. Bottom shows masked version of MSCL (using mask shown in figure 5.1). Left column uses constant magnitude function equal to one. Right column uses as magnitude function the absolute value of first component of each sample.

training). Differences with MSCL are due to the pulling effect of the MS-SOM.

These results demonstrate that the new algorithm is able to deal with incomplete data in its two versions, using a MSCL neural network and a MS-SOM neural network.

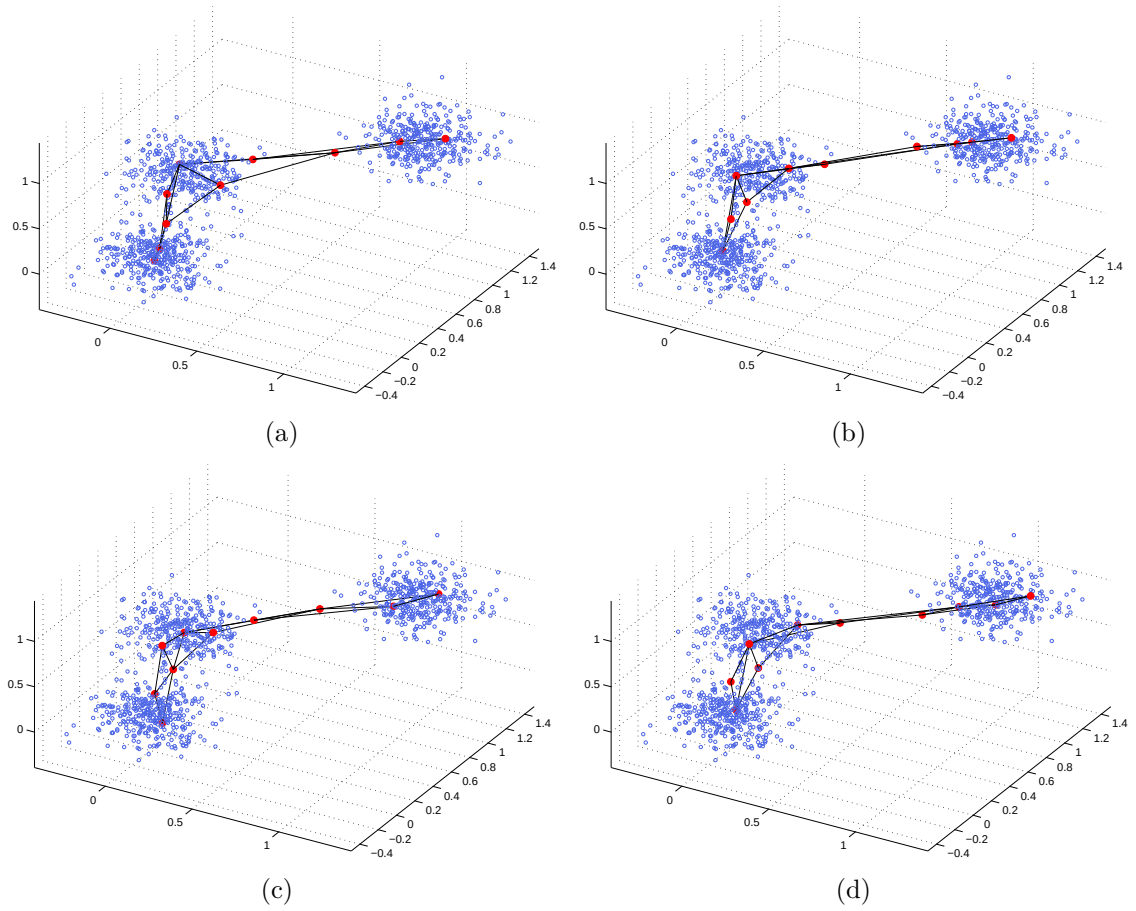


Figure 5.3: Final results of a trained MS-SOM with 10 units using two magnitude functions. On top, row figures represent direct training without taking the mask into consideration. Bottom shows masked version of MS-SOM (using mask shown in figure 5.1). Left column uses constant magnitude function equal to one. Right column uses as magnitude function the absolute value of first component of each sample.

Chapter 6

Magnitude Sensitive Initialization

6.1 Introduction

As it has be mentioned in chapter 2, one of the most popular clustering algorithms is the K-means algorithm. This algorithm is a greedy algorithm for minimizing the MSE error, hence, it may not converge to the global optimum. The performance of K-means strongly depends on the initial guess of partition (that is, in the selection of initial centroids). To achieve a good initialization, many other techniques have been proposed apart of random initialization with data samples.

For instance, Cutting [19] used group average agglomerative clustering to select initial centroids. Likas [44] proposed the global K-Means algorithm, an incremental approach to clustering which dynamically adds one cluster center at a time through a global search consisting on several executions of the K-Means algorithm. Onoda [57] proposed a seeding method based on the independent component analysis. Katsavounidis et. al [36] proposed a method, KKZ, that utilizes the sorted pairwise distances for initialization.

Some of these algorithms are not viable in practice as they are computationally exponential in K. In 2007 Arthur and Vassilvitskii [8] proposed the K-means++ algorithm. This algorithm chooses new centers by weighting of data points according to their squared distance from the closest center already chosen. This algorithm improves both speed and efficiency of K-means.

In this chapter we present a new algorithm, *Magnitude Sensitive Init* (MS-INIT) as a generalization of both K-Means++ and KKZ when data samples are additionally weighted with any kind of magnitude. This algorithm chooses centroids such they are focused in space areas with high magnitude. Therefore, it is a good initialization method for MSCL,

as it will be demonstrated later.

The chapter has three sections. First, K-Means++ and KKZ are outlined. Then, we present the new MS-INIT algorithm. Finally, some experiments show the algorithm capabilities.

6.2 Related algorithms: K-means++ and KKZ

6.2.1 KKZ

The KKZ algorithm works as follows:

1. The codebook set is initialized to contain only one unit, with codeword $\mathbf{w}_1$ randomly selected from the dataset $\mathcal{X}$:

$$\mathcal{M} = \{u_1\} \tag{6.1}$$

2. Calculate for each input sample $\mathbf{x} \in \mathcal{X}$ the value of $d(\mathbf{x})$ as the shortest distance from a data point to the closest of centers already chosen:

$$\begin{aligned} j &= \underset{m \in \mathcal{M}, \mathbf{x} \in \mathcal{X}}{\operatorname{argmin}} (\|\mathbf{x} - \mathbf{w}_m\|) \\ d(\mathbf{x}) &= \|\mathbf{x} - \mathbf{w}_j\| \end{aligned} \tag{6.2}$$

3. Take a new center u_n, choosing $\mathbf{w}_n = \mathbf{x}$ the sample with maximum value of d.

$$\begin{aligned} u_n &= \underset{\mathbf{x} \in \mathcal{X}}{\operatorname{argmax}}(d(\mathbf{x})) \\ \mathcal{M} &= \mathcal{M} \cup \{u_n\} \end{aligned} \tag{6.3}$$

4. Repeat step 2 until we have chosen M centers altogether (we say that we choose "M" centers instead of "K" for homogeneity in notation with other parts of this Thesis). Once the algorithm has finished its processing, this is the codebook set:

$$\mathcal{M} = \{u_1, u_2, \cdots, u_M\} \tag{6.4}$$

6.2.2 K-means++

K-means++ works as follows:

1. The codebook set is initialized to contain only one unit, with codeword $\mathbf{w}_1$ randomly selected from the dataset $\mathcal{X}$:

$$\mathcal{M} = \{u_1\} \tag{6.5}$$

2. Calculate for each input sample $\mathbf{x} \in \mathcal{X}$ the value of $d(\mathbf{x})$ as the shortest distance from a data point to the closest of centers already chosen:

$$\begin{aligned} j &= \underset{u_m \in \mathcal{M}, \mathbf{x} \in \mathcal{X}}{\operatorname{argmin}} (\|\mathbf{x} - \mathbf{w}_m\|) \\ d(\mathbf{x}) &= \|\mathbf{x} - \mathbf{w}_j\| \end{aligned} \tag{6.6}$$

3. Take a new center u_n, choosing $\mathbf{w}_n = \mathbf{x}$ with probability

$$p(\mathbf{x}) = \frac{d(\mathbf{x})}{\sum\limits_{\mathbf{x} \in \mathcal{X}} d(\mathbf{x})} \tag{6.7}$$

$$\mathcal{M} = \mathcal{M} \cup \{u_n\} \tag{6.8}$$

4. Repeat step 2 until we have chosen M centers altogether. Once the algorithm has finished its processing,

$$\mathcal{M} = \{u_1, u_2, \cdots, u_M\} \tag{6.9}$$

6.3 MS-Init

The new algorithm, MS-INIT requires the definition of the value of a magnitude mx associated to each sample $\mathbf{x}$. The algorithm works as follows:

1. The codebook set is initialized to contain only one unit, with codeword $\mathbf{w}_1$ randomly selected from the dataset $\mathcal{X}$ according to the probability of appearance of $\mathbf{x}$:

$$p(\mathbf{x}) = \frac{mx}{\sum\limits_{\mathbf{x} \in \mathcal{D}} mx} \tag{6.10}$$

2. Calculate for each input sample $\mathbf{x} \in \mathcal{X}$ the value of $d(\mathbf{x})$ as the shortest distance from a data point to the closest of centers already chosen:

$$j = \underset{u_m \in \mathcal{M}, \mathbf{x} \in \mathcal{X}}{\operatorname{argmin}} \|\mathbf{x} - \mathbf{w}_m\|$$
$$d(\mathbf{x}) = \|\mathbf{x} - \mathbf{w}_j\| \tag{6.11}$$

3. Choose a new center u_n, making $\mathbf{w}_n = \mathbf{x}$, with one of this two possibilities, following the probability of the product of the magnitude associated to each sample $mx(\mathbf{x}) \cdot d(\mathbf{x})$ (eq. 6.12a) or maximizing this product (eq. 6.12b) :

$$p(\mathbf{x}) = \frac{mx(\mathbf{x}) \cdot d(\mathbf{x})}{\sum\limits_{\mathbf{x} \in \mathcal{X}} mx(\mathbf{x}) \cdot d(\mathbf{x})} \tag{6.12a}$$

$$u_n = \underset{\mathbf{x} \in \mathcal{X}}{\operatorname{argmax}}(mx(\mathbf{x}) \cdot d(\mathbf{x})) \tag{6.12b}$$

4. Add u_n to the codebook set ($\mathcal{M} = \mathcal{M} \cup \{u_n\}$) and repeat step 2 until we have chosen K centers altogether. Once the algorithm has finished its processing, the codebook is:

$$\mathcal{M} = \{u_1, u_2, \cdots, u_M\} \tag{6.13}$$

The algorithm produces a codebook initialization with unit prototypes focused in zones where mx is high, as it will be shown in the experimental section. Obviously if the value of the magnitude is constant and equal to 1, equation (6.12a) is the same as the equivalent in K-means++ and eq. (6.12b) as the KKZ algorithm. Therefore, MS-INIT is a generalization of both algorithms, when magnitude information is provided.

6.4 Experiments

The aim of these experiments is to demonstrate that the MS-INIT algorithm is capable of initializing a codebook of a desired size from a dataset $\mathcal{X}$, following a defined magnitude.

The selected dataset is the 3 Gaussian distributions used in other chapters (i.e, in subsection [4.3.1]). This dataset is represented in figure 6.1(a) were each sample has a colour code according to its magnitude: black color means magnitude near zero, while higher magnitude is represented in red.

Each sample has a associated value of magnitude given by:

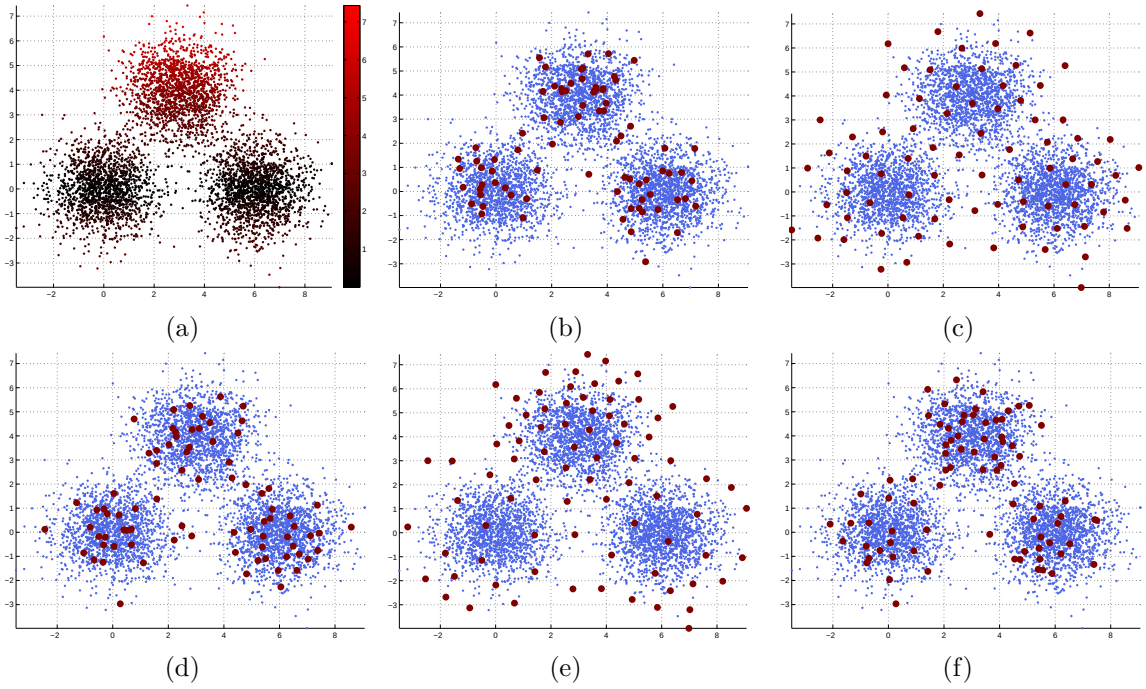


Figure 6.1: (a) Dataset used in the examples of this chapter (with color coding according to the magnitude associated to each sample) and codebooks of 80 units initialized with the different methods: (b) Random init, (c) KKZ algorithm, (d) Kmeans++ algorithm, (e) MS-INIT$_{max}$ algorithm and (f) MS-INIT$_{prob}$ algorithm.

$$mx(\mathbf{x}) = abs(x_2) \tag{6.14}$$

Figure 6.1$(b)-(f)$ shows the results of initializing a codebook of 80 units with the five methods. In blue it is represented the dataset, while unit centers are the coloured circles.

6.4.1 Initialization example

In this example, each test consists on the initialization of three codebooks of sizes M = [40,80,160] using the following methods for initialization:

- *Random init*: Directly M samples are selected at random to form the codebook.
- *KKZ*: Implemented with MS-INIT using magnitude = 1, maximizing $d(\mathbf{x})$.
- *K-Means++*: Implemented with MS-INIT using magnitude = 1 and following the probability of $d(\mathbf{x})$

- *MS-INIT*$_{max}$: Following equation (6.12b). Therefore, maximizing the product $mx(\mathbf{x})\cdot d(\mathbf{x})$.

- *MS-INIT*$_{prob}$: Following the probability of the product $mx(\mathbf{x}) \cdot d(\mathbf{x})$, as indicated in eq. (6.12a).

Since all algorithms under test select first unit randomly, what could affect the final result, we ran 100 trials for every case. As a measure of performance quality of the new method we use magnitude-weighted versions of MSE, and normalized Entropy. Table 6.1 shows the means (in 100 trials) of weigthed MSE, while table 6.2 shows the means of weigthed entropy. By rows are ordered three different codebook lengths: 40, 80, and 160 units, and by columns shows results of applying the five initialization algorithms. Every three possible training schemes after each initialization are marked with: None, K-means and MSCL. Results of training the neural network after initialization (with different methods) are explained in next section.

The number of units affects quantization in a inverse proportion and entropy in a direct way. Higher number of units means lower value for the Weighted MSE and higher Weighted Entropy (that tends to a value of one when normalized). For the None situation in the tables, the representation is obtained with the initialization method without training.

Comparing these values the initialization algorithm that yields worse results is *random init* as expected (except in the case of low number of units where *KKZ* forces units to take the value of extreme samples in the dataset and therefore the weighted quantization error increases). *KKZ* tends to distribute codewords uniformly in the data space, so becoming very expansive (some of the reference vectors are in the limits of the three Gaussian distributions). *K-means++* also distributes codewords at random but with a unit distribution that tends to follow the data density distribution. However, all of these methods are still worse than MS-INIT when the goal is reducing the quantization error following a magnitude function.

When goal is achieving high entropy, worse results are for KKZ and MS-INIT$_{max}$ as none of these two algorithms achieve an uniform probability distribution (as it can be seen in table 6.2 in the rows of training equal to 'None').

In general, implementations of MS-INIT achieve better results, but it is clear that using MS-INIT$_{prob}$ is advantageous against the use of MS-INIT$_{max}$ regarding the performance results. In figure 6.1 it can be seen that few more units are distributed in the limits of Gaussian dataset with highest magnitude than in the case of MS-INIT$_{prob}$.

Units	*Training*	Random	KKZ	Kmeans++	MSINIT$_{max}$	MSINIT$_{prob}$
40	*None*	0.678 (0.049)	*0.757 (0.038)*	0.621 (0.031)	0.655 (0.030)	0.578 (0.019)
	Kmeans	0.535 (0.024)	0.525 (0.010)	0.528 (0.020)	0.496 (0.006)	0.499 (0.010)
	MSCL	0.475 (0.010)	0.474 (0.007)	0.470 (0.007)	**0.464 (0.006)**	0.468 (0.007)
80	*None*	*0.485 (0.039)*	0.468 (0.017)	0.433 (0.016)	0.417 (0.012)	0.392 (0.015)
	Kmeans	0.371 (0.012)	0.365 (0.005)	0.367 (0.011)	0.349 (0.004)	0.339 (0.006)
	MSCL	0.336 (0.006)	0.337 (0.004)	0.334 (0.006)	0.325 (0.003)	**0.322 (0.003)**
160	*None*	*0.342 (0.016)*	0.300 (0.006)	0.297 (0.010)	0.270 (0.004)	0.267 (0.005)
	Kmeans	0.245 (0.009)	0.251 (0.004)	0.243 (0.008)	0.229 (0.002)	0.224 (0.003)
	MSCL	0.231 (0.007)	0.233 (0.002)	0.231 (0.005)	0.220 (0.001)	**0.217 (0.002)**

Table 6.1: Table shows the mean values in 100 tests of the *Weighted Mean Square Error (WMSE)* calculated in three codebooks (sizes 40, 80 and 160) after applying five initialization algorithms (*Random, KKZ, Kmeans++, MSINIT$_{max}$ and MSINIT$_{prob}$*). Each of these codebooks is trained following one of three possibilites: *No training* / Trained using *Kmeans* / Trained using *MSCL* (with the same magnitude funcion as for MS-INIT).

Units	*Training*	Random	KKZ	Kmeans++	MSINIT$_{max}$	MSINIT$_{prob}$
40	*None*	*0.857 (0.035)*	0.752 (0.012)	0.874 (0.021)	0.804 (0.019)	0.914 (0.016)
	Kmeans	0.890 (0.022)	0.893 (0.007)	0.896 (0.017)	0.927 (0.006)	0.928(0.010)
	MSCL	0.956 (0.010)	0.954 (0.004)	0.958 (0.008)	**0.970 (0.004)**	**0.970 (0.006)**
80	*None*	*0.878 (0.027)*	0.804 (0.011)	0.890 (0.014)	0.849 (0.010)	0.940 (0.010)
	Kmeans	0.900 (0.017)	0.892 (0.006)	0.902 (0.014)	0.925 (0.004)	0.946 (0.009)
	MSCL	0.948 (0.009)	0.949 (0.005)	0.951 (0.007)	**0.976 (0.002)**	0.974 (0.006)
160	*None*	*0.900 (0.016)*	0.841 (0.006)	0.904 (0.010)	0.886 (0.004)	0.945 (0.005)
	Kmeans	0.917 (0.009)	0.885 (0.004)	0.913 (0.008)	0.927 (0.002)	0.952 (0.003)
	MSCL	0.944 (0.007)	0.942 (0.002)	0.944 (0.005)	0.968 (0.001)	**0.971 (0.002)**

Table 6.2: Table shows the mean values in 100 tests of the normalized *Weighted Entropy* calculated in three codebooks (sizes 40, 80 and 160) after applying five initialization algorithms (*Random, KKZ, Kmeans++, MSINIT$_{max}$ and MSINIT$_{prob}$*). Each of these codebooks is trained following one of three possibilites: *No training* / Trained using *Kmeans* / Trained using *MSCL* (with the same magnitude funcion as for MS-INIT).

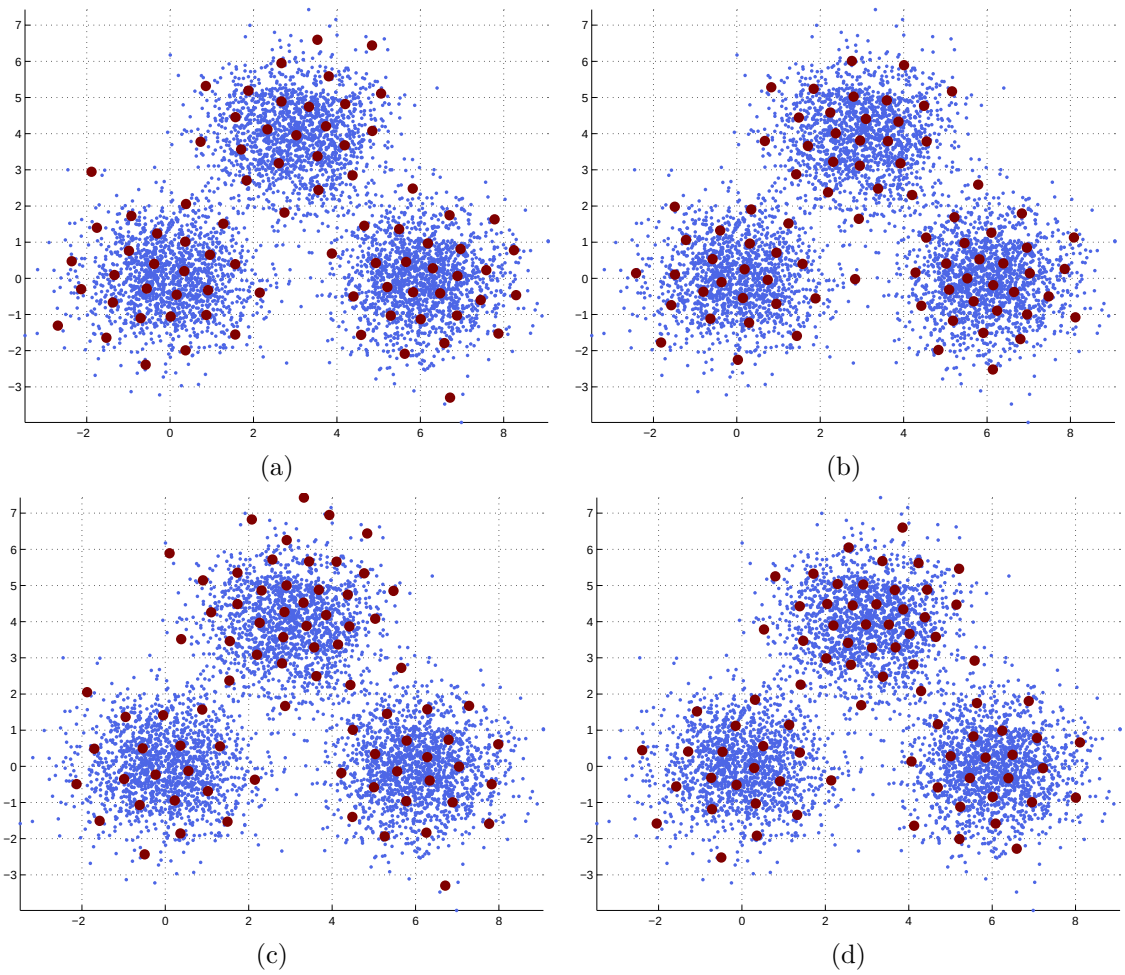


Figure 6.2: Final results after training with K-means a codebook with 80 units, using different initialization methods: (*a*) KKZ algorithm, (*b*) Kmeans++ algorithm, (*c*) MS-INIT$_{max}$ algorithm and (*d*) MS-INIT$_{prob}$ algorithm.

6.4.2 Training example

Next example has been developed to show the effect of different initialization schemes in the final codebook of a competitive neural network after training. To do it we apply the initialization methods used in the previous example to three *MSCLs* of different size (also 40, 80 and 160 units) and to three *K-Means* neural networks, with the same number of clusters, K. Performance measures will be the same than in the initialization example.

Training is made in batch mode in all the cases using:

- *K-Means*: Repeating training during 10 cycles.

- *MSCL*: Training each neural network during 10 cycles, and 5 epochs each cycle. Selected magnitude function will be the same than the used for MS-INIT.

Figure 6.2 shows final training results of 80 centroids using K-means following the initialization methods: KKZ (a), K-Means++ (b), MS-INIT$_{max}$ (c) and MS-INIT$_{prob}$ (d). Figure 6.3 shows final training codebook of the MSCL with 80 units, being initialized as in figure 6.2.

Tables 6.1 and 6.2 also show the quality results after training the with *K-means* neural networks and the *MSCLs* respectively. As in the previous example, neural networks with more units present better results. MS-INIT has lower values in WMSE and higher Weighted Entropy than the other three initialization algorithms. Once again, KKZ and MS-INIT$_{max}$ have lower values for entropy due to the fact that both algorithms tends to select some of the unit weights from data samples in the limits of the distributions.

Obviously, as the selected magnitude function for training the MSCL is the same than in the MS-INIT method, MSCL gets better final center representations than K-Means. Finally it is worth noting the fact that optimal values are achieved using MS-INIT and then training a MSCL with the same magnitude function. Besides, it is preferable training a MSCL while using random initialization, than only initializing the codebook with MS-INIT and performing no subsequent training.

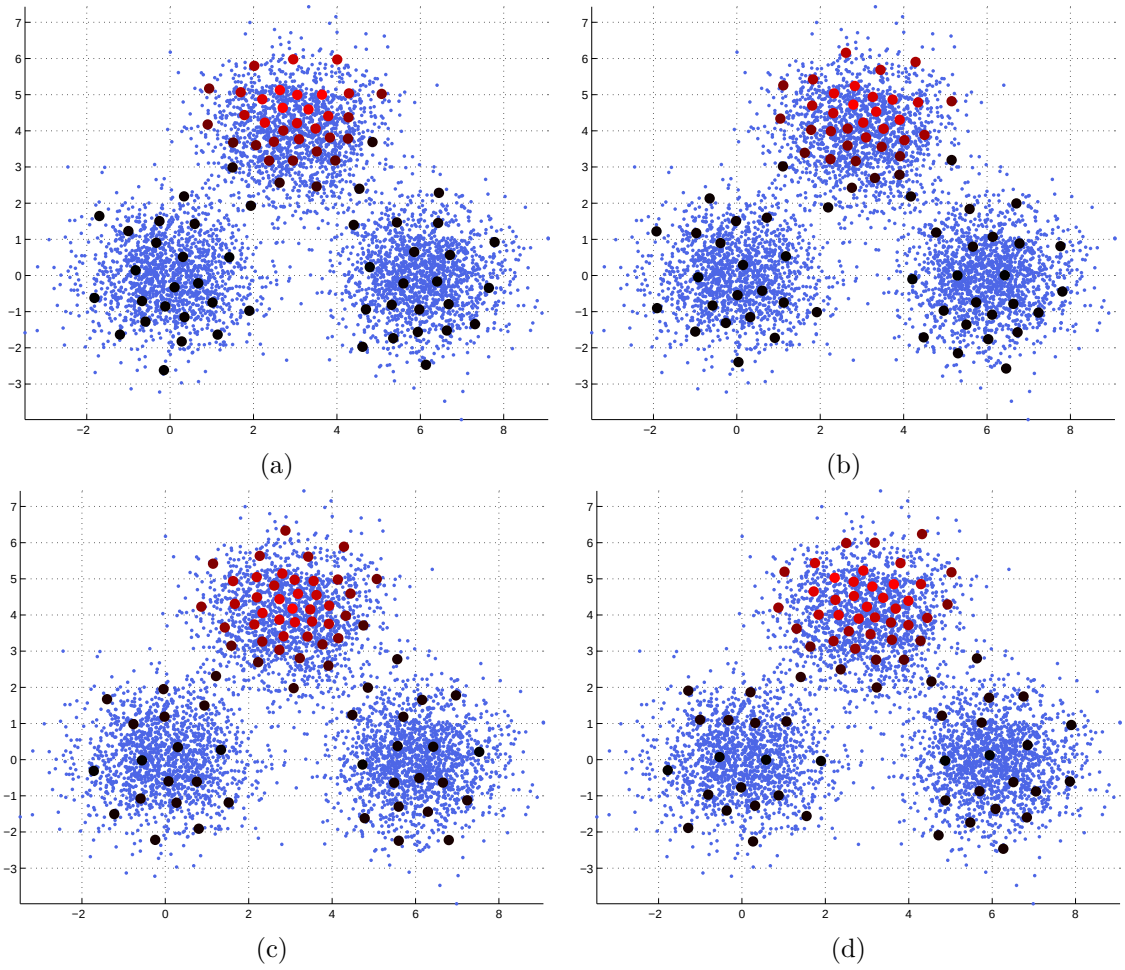


Figure 6.3: Final results after training a MSCL with 80 units, using different initialization methods: (*a*) KKZ algorithm, (*b*) Kmeans++ algorithm, (*c*) MS-INIT$_{max}$ algorithm and (*d*) MS-INIT$_{prob}$ algorithm.

Part III

APPLICATIONS

Chapter 7

Color Quantization with MSCL

7.1 Introduction

With the informatics development of society, large amount of scanned documents and images are transmitted and stored, therefore it would be desirable to reduce the number of colors in an image to reduce storage and transmission costs. This is generally achieved by mean of Vector Quantization techniques called Color Quantization (CQ) where each data sample is a vector representing the color of a pixel. They are important in certain applications related to segmentation, compression, and transmission of images.

Some of the most common competitive learning methods, or their variants, have already been used in CQ and Color Segmentation tasks. Uchiyama and Arbib [75] developed Adaptive Distributing Units (ADU), a CL algorithm used in Color Segmentation that is based on a simple cluster splitting rule. More recently, Celebi [12] demonstrated that this algorithm outperforms other common algorithms in a CQ task. Fuzzy C-Means (FCM), is a well-known clustering method in which each sample can belong to more than one cluster [10]. In [13], Celebi presented a relevant work using Neural Gas Networks.

SOM has also been used in color related applications: in binarization [59], segmentation [43] and CQ [21], [56], [17] and [16] where author presents FS-SOM a frequency sensitive learning scheme including neighborhood adaptation that achieves similar results to SOM, but is less sensitive to the training parameters. One variant of special interest is the neural network Self-Growing and Self-Organized Neural Gas (SGONG) [9], an hybrid algorithm using the GNG mechanism for growing the neural lattice and the SOM leaning adaptation mechanism. Author proved that it is one of the most efficient Color Reduction algorithms, closely followed by SOM and FCM.

In this chapter we propose the use of a MSCL neural network in the CQ task. As a result of the training process, units will distribute according to the salient pixels in the image, where different definitions of saliency are used as magnitude functions. Final color palette will therefore enhance salient areas of the image.

7.1.1 Problem formulation

Given an image I of size (x_{max}, y_{max}), we define a data sample $\mathbf{x}(t)$ from the pixel I(x,y) as the color vector of that pixel in the coordinates in the corresponding color space:

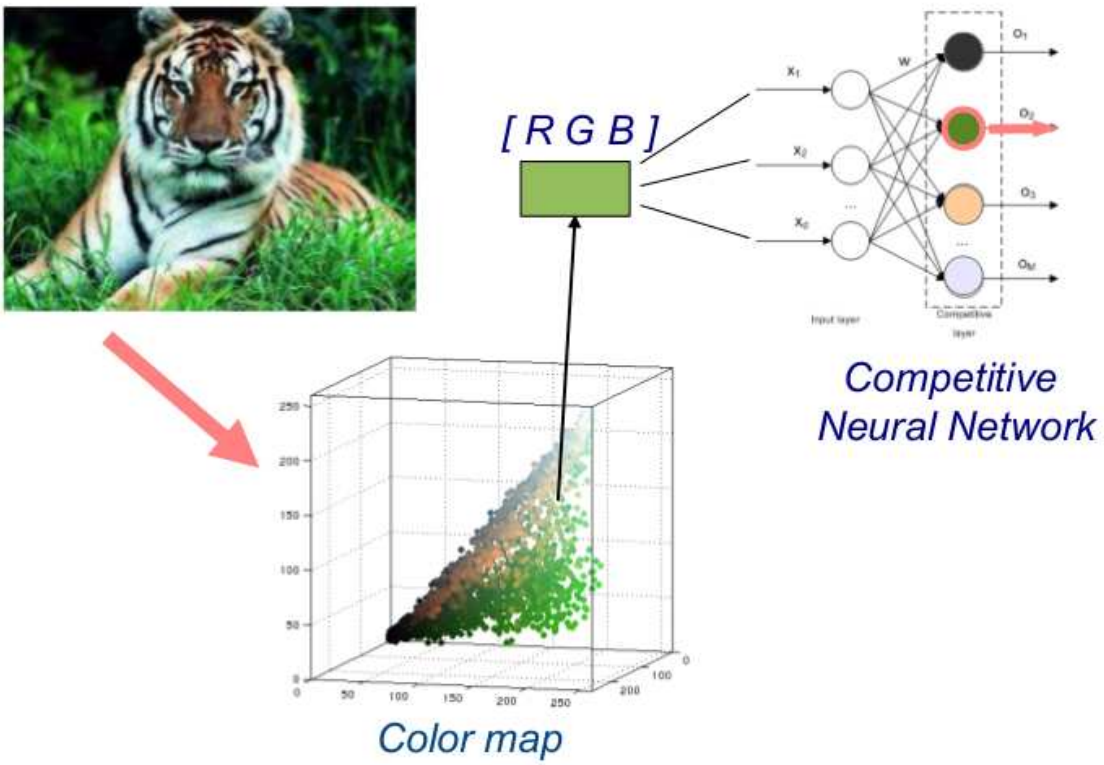


Figure 7.1: Problem formulation of Color Quantization: pixels are considered 3-dimensional vectors that are processed as inputs for a competitive neural network with many units as colors in the palette. Magnitude value can be associated to the pixel, as another input to the network, or be associated to the units, as an internal parameter.

$$\mathbf{x}(t) = Color(I(x,y)), \quad x = 1..x_{max} \quad \text{and} \quad y = 1..y_{max} \tag{7.1}$$

$$t = 1..P, \quad \text{where} \quad P = x_{max} * y_{max} \ . \tag{7.2}$$

Each pixel receives an additional value for the magnitude function, $MF(t)$. This function is proposed as to weight each pixel with a value of interest. As higher is the value of magnitude, more interesting is the pixel. The goal is training a neural network to get units representing the colors of the interesting pixels in higher detail. The prototype of unit m ($m = 1 \ldots M$) is represented by a vector of weights in the 3-dimensional color space. The associated value of the magnitude in that unit, $mu_m(t)$, can be calculated from the values of $MF(t)$ at samples in its Voronoi region, or can be introduced to the network as the rest of input data.

Figure 7.1 shows this process. From one image (tiger example), we get a dataset with the 3-D color coordinates of each pixel. The dataset is presented to the competitive neural network to generate the color palette with as many colors as units in the network. In the bottom of the image, the color distribution of the image using a [R G B] color space is shown.

7.1.2 Proposed approach

We propose the use of MSCL neural network, to train this 3-D dataset, taking into account the magnitude function, which can be defined to lead the training process of the palette to accomplish any desired task.

7.1.3 Chapter description

The remainder of this chapter is organized as follows:

Section 7.2 shows the comparison of the MSCL with some of the well known competitive learning algorithms, using five different examples of applications. The first example gets a color quantization that we call homogeneous quantization (Subsect. 7.2.1). The second example gets a color quantization focused in the image center (Subsect. 7.2.2). The third example is focused on getting a color palette avoiding the dominant colors usually found in image background (Subsect. 7.2.3). Fourth example returns a color palette according with a certain image saliency (Subsect. 7.2.4). Last example shows the use of MSCL in a document image binarization (Subsec. 7.2.5).

7.2 Applications

In the examples, data samples are 3D vectors corresponding to the RGB components of the image pixels. We have used the RGB space in order to have comparable results to other works, in spite that it is a non-uniform color space (instead of using this one, we could have used other color models as $L * a * b$ whose suitability has been demonstrated for interpreting the real world).

The goal is to get a reduced color palette to represent the colors in the image focused on different objectives. The next five examples show that, adequately selecting the magnitude function, it is possible to get an optimal palette according to the desired application.

Figure 7.2: Original Tiger image (*top-left*) and its reconstruction using 8 colors applying: ADU (*top-right*), Homogeneous MSCL (*bottom-left*) and Centered MSCL (*bottom-right*).

Pixels	*Image*	Som	FSCL	*M-h*	FCM	FSSom	ADU	*M-c*	Sgong
Whole img.	*T8*	987	1016	1037	1005	**985**	990	1095	987
	T16	566	596	577	606	564	**562**	667	570
	T32	334	343	341	357	328.1	**327.8**	409	574
	L8	401	416	424	451	**400.2**	406	406	400.9
	L16	216	234	215	234	216	**214**	217	218
	L32	121	126	122	141	120	**119**	125	222
	B8	1120	1126	1138	1151	**1117**	1126	1227	1121
	B16	633	641	633	693	**632.4**	632.8	751	635
	B32	380	389	380	440	**375.2**	375.9	479	442
Img. center	*T8*	1223	1311	1207	1263	1214	1244	**1151**	1226
	T16	626	710	596	735	631	608	**485**	655
	T32	361	381	356	408	353	355	**283**	407
	L8	445	472	436	552	440	447	**423**	447
	L16	265	294	273	301	262	266	**254**	267
	L32	161	167	160	187	159	159	**149**	163
	B8	1346	1354	1210	1421	1343	1338	**1062**	1321
	B16	708	740	683	833	705	689	**602**	714
	B32	381	412	387	515	372	374	**354**	539

Table 7.1: MSE calculated in the whole image and in the image center.

7.2.1 Homogeneous color quantization

This example shows the case we call Homogeneous Color Quantization. The mean quantization error (q_{err}) for all samples within the Voronoi region of unit i is used as magnitude function. The q_{err} of a sample $\mathbf{x}(t)$ is the distance between $\mathbf{x}(t)$ and the prototype (weights) of its corresponding best matching unit. This magnitude forces the palette colors to be uniformly distributed over the dataset in the RGB space, independently of its data density, and resulting in Voronoi regions with similar mean q_{err}.

We use the known Tiger, Lena and Baboon images for performance comparison in the CQ task (marked in the table as *T**, *L** and *B**, where * is the number of colors). Homogeneous MSCL (M-h) and Centered MSCL (M-c, explained in next subsection) are compared against the most successful neural models used in different papers: SOM, FSCL, FCM, FS-SOM, ADU and SGONG. Training process applied learning rates between (0.7-0.01) along three cycles, except in ADU whose algorithm parameters selection follows [75]. The threshold for adding/removing a neuron used in SGONG was (0.1/0.05).

Figure 7.2 shows the color reduction effects for tiger image with ADU, Homogeneous

MSCL and Centered MSCL. The upper part of Table 7.1 shows the mean of MSE (Mean Squared Error) in 10 trials with different number of palette colors (8, 16 and 32) calculated in the whole image. Peak Signal-to-Noise Ratio (PSNR) measure can be easily calculated from MSE value. In general, ADU outperforms all other models, closely followed by SOM and FS-SOM. However, it is clear that ADU (top-right image in Fig. 7.2) paints the tiger skin with greenish color as an effect of the over-representation of green colors. Both MSCL results (bottom images in Fig. 7.2) tend to maintain orange colors in the tiger skin, as they are not focused in data density representation.

7.2.2 CQ Focused on the image center

Previous example provides a CQ task giving equal importance to every pixel of the image, and not distinguishing between pixels from the foreground or the background. However the more interesting image regions are usually located in the foreground center. Using MSCL with the adequate magnitude function, it is possible to get a palette with colors mainly adapted to pixels located in the foreground, or any other desired point in the image. In this example we use the following magnitude function for each sample:

$$MF(t) = 1 - d(\mathbf{x}(t)) \tag{7.3}$$

where $d(\mathbf{x}(t))$ is the normalized distance, in the plane of the image (x, y), calculated from the corresponding pixel position to the center of the image. This magnitude function is normalized by the maximum.

We compare the performance of centered MSCL, with the same methods used in previous example. Number of colors and training parameters were also the same.

As it can be seen in the lower part of Table 7.1, prototypes of centered MSCL tend to focus on colors in the central part of the image, so the MSE for the whole image is worse than those obtained using other methods, as background is under-represented. However, when repeating the measures in the central area of the image (150x170 pixels), this algorithm (column *M-c* in the table) outperforms the others, because its color palette models with more detail the central region of the image.

7.2.3 CQ Avoiding dominant colors

Many natural images present few dominant background colors. It means that the majority of the image pixels are represented with a limited set of colors, while other small chunks of the image use a wider palette. That is why, when it is applied traditional Competitive

Learning algorithms for color quantization on this kind of images, final color palette usually over-represents dominant colors, and other secondary colors tend to be dismissed.

In this example MSCL is used to get a reduced color palette avoiding the color dominance. This goal is accomplished in a two-step method. First the dominant colors of the image are found, second, MSCL is applied to avoid these dominant colors by defining a magnitude function that gives higher values to the pixels that are more distant from them. We tested this methodology with 4 images (shown in top of Fig. 7.3: fish, flower, tower, goat) and compared it with the results of 5 neural models used in different papers: FSCL, FCM, Neural Gas (NG), K-Means and SOM.

Following we describe the two-step method in detail and the results of the experiments.

Figure 7.3: Original images used in the example of MSCL avoiding dominant colors and one example of the corresponding dominant color palettes (from 1 to 8 colors).

Determination of Dominant Colors.

First step is the determination of the dominant colors in the image. One simple way to do it, frequently used in the literature, is using some type of competitive learning algorithm to cluster pixel colors. Weights of units after training the neural network will be the dominant colors. A good candidate for this approach is FSCL method. FSCL can be considered a particular case of MSCL where $MF(t)$ for sample $\mathbf{x}(t)$ is the number of hits of its best matching unit. We use this definition to implement FSCL as a data-density sensitive method that is able to cluster data colors.

It would possible to use a unique simulation of FSCL to obtain the dominant colors. However, this method is dependent of the goodness of the unit initialization. So, in order to smooth this 'noisy' initialization in the results of the analysis, we use an ensemble of 50 FSCLs for each number of dominant colors (except for the case of one dominant color, calculated as the mean of the image colors). After generating the 50 networks of each ensemble, their prototypes are used to train the final FSCL to get the 'averaged' dominant colors. We call $\mathbf{pal}_k$ with $k \in \{dominants\}$ to the final dominant-colors palette.

Bottom of Fig. 7.3 shows the resulting palettes from 1 to 8 dominant colors in the four test images. The evolution of these palettes show that, in the fish dominant-color palette, orange does not appear until using 5 dominant colors. The flower needs 4 dominant colors to show a good red color, and the tower needs 8 dominant colors to include the red in the roof. The goat dominant-color palette shows that the palette is quite monochromatic.

MSCL Avoiding Dominant Colors.

The magnitude function $MF(t)$ used in this example needs to exhibit higher values as the pixel color is more distant from the dominant-color palette. So, for each palette of dominant colors, we define a function ($distcol(t)$) for each pixel as the distance in the color space from that pixel to the closest color in the $\mathbf{pal}_j(t)$ palette:

$$j = \underset{k}{\operatorname{argmin}}(\|\mathbf{pal}_k(t) - \mathbf{x}(t)\|) \quad k \in \{dominants\} \tag{7.4}$$

$$distcol(t) = \|\mathbf{pal}_j(t) - \mathbf{x}(t)\| \ . \tag{7.5}$$

Figure 7.4 shows how this magnitude function works in the case of the fish image using an 8-color palette that avoids two dominant colors. A fraction of the pixels in the color distribution is depicted jointly with the closest regions of the dominant colors (large red circles) and the prototypes generated with MSCL (8 blue circles). MSCL uses three

palette-colors for the orange colors of the fish, two colors for the white tones, one stronger black and only two colors dedicated to the background colors with the anemone. One of these colors almost matches with one of the dominant color (in the center of the graph). This result comes out because there is a large amount of the pixels in this zone, and MSCL is forced to move a prototype to this zone to reduce quantization error.

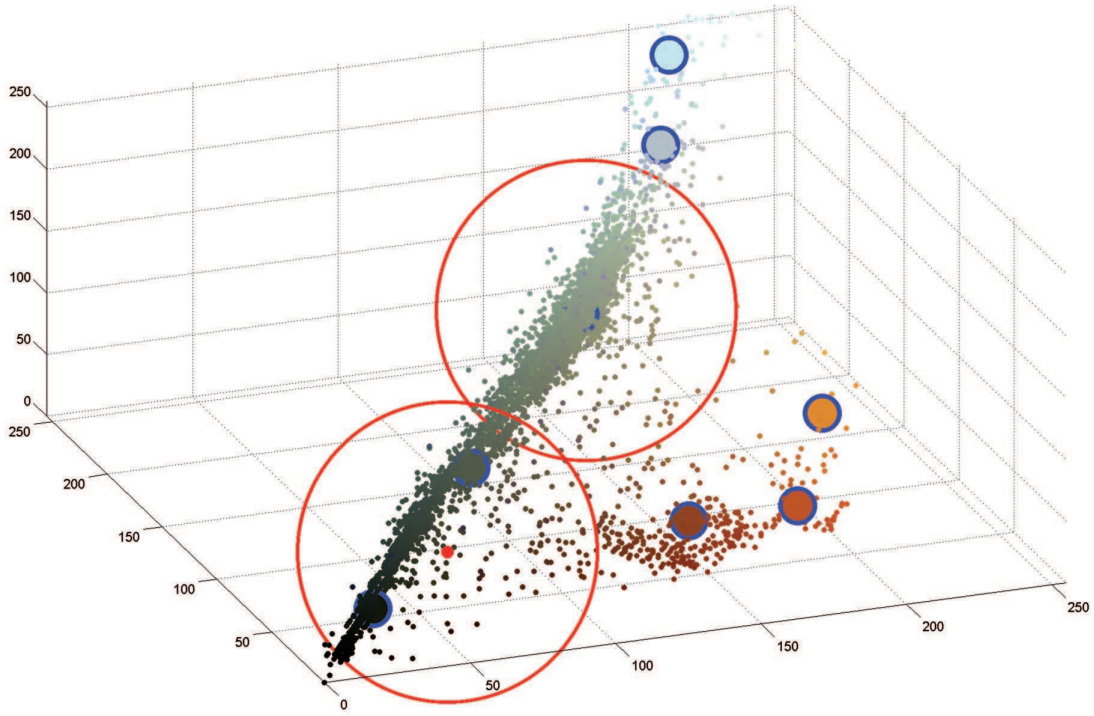


Figure 7.4: Representation of a fraction of the pixels in the color distribution for the fish image. The large red circles represent the regions close to the two dominant colors of the image. The 8 blue circles represent the 8-color palette obtained for MSCL avoiding those dominant colors. MSCL uses three palette-colors for the orange colors of the fish, two colors for the white tones, and only three colors dedicated to the background colors.

In Fig. 7.5 the reconstructed images with 8-color palettes of this image are shown, from left to right and top to bottom: MSCL avoiding two dominant colors, NG, FSCL, FCM, K-MEANS and SOM. It can be appreciated that MSCL obtained a more vivid color representation for the fish, losing the detail in the anemonae, while other algorithms tend to concentrate the units in the most common colors, showing a lot of greyish tones of the anemone.

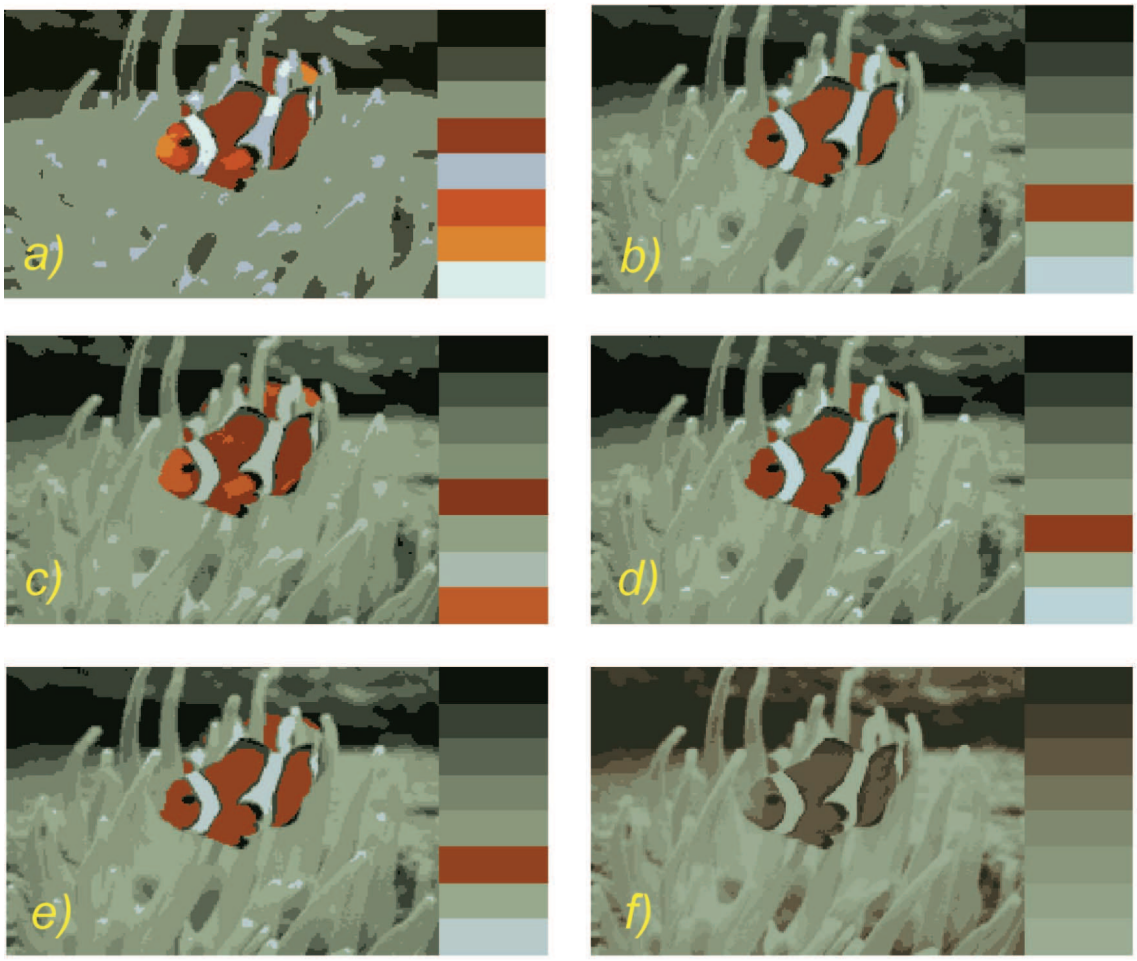


Figure 7.5: Results of color quantization for the fish example using an 8-color palette with different methods: *a)* MSCL avoiding two dominant colors, *b)* NG, *c)* FSCL, *d)* FCM, *e)* K-MEANS, *f)* SOM. The corresponding color palettes are shown in the right of each image. As can be appreciated, MSCL gets a more vivid palette for the fish and presents a lower number of colors in the palette dedicated to the background with the anemone.

Results of experiments avoiding Dominant Colors.

The main problem of the method is to determine the optimum number of dominant colors. So, we propose to calculate the amount of pixels in High Magnitude Regions (HMR) as a measure of the level of detail that the MSCL method has to deal with. The HMR in the image can be estimated with a threshold of the magnitude function that is chosen to be the 50% of the maximum magnitude value. Figure 7.9 shows this process for the fish example in an 8-color palette. Images from top to bottom in each column correspond to number

of dominant colors from 1 to 8. The first column represents the value of $distcol(t)$ in form of image (the magnitude map). The second column shows the HMR as the corresponding binarization of the magnitude maps. It is worth noting in this col that there are still quite a lot of pixels in HMR considering only one dominant color (corresponding to the fish and the darker areas in the background of the image). However, when two dominant colors are used, HMR extension is quite reduced and corresponds only to certain areas in the fish. Therefore the use of two dominant colors would be a good option for the fish image. The third column of images in this figure shows examples of the MSCL reconstruction for the corresponding number of dominant-colors avoidance.

As comparison, Fig. 7.10 shows in three columns the resulting HMRs for the other three images. In the first column, the flower image presents an interesting behavior for four dominant colors. In the tower image we have a similar situation, but for three colors (white, blue and dark grey). However, the goat image tends to keep similar HMR extensions. The most possible reason for this behavior is that the image is quite monochromatic.

In order to visualize the effect of the number of dominant colors, we define the HMR-ratio as the number of pixels in HMR divided by the number of pixels in the image. We generated 50 palettes of dominant colors for each number of colors that varied from 1 to 20. The evolution of the averaged HMR-ratios are shown if the bottom graph of Fig. 7.8. The curves in the graphs have been smothered. The abscissa shows the different number of dominant colors analysed in the four images. The ordinates show the mean value of the HMR-ratio. A lower value in this ratio means that there are fewer pixels in the image far from the dominant colors. Therefore that palette is a good representative of the dominant colors in the image.

The evolution of HMR for the fish image shows that there is an abrupt fall in this value from using one or two dominant colors. This ratio tends to keep consistent until 7 dominant colors are used. An explanation of this behavior can be visualized in Fig. 7.9 (image in row 7 and second column) where the dark band in the background is far from any dominant color, which makes the HMR-ratio to grow considerably in the bottom graph of Fig. 7.8.

It would be possible to detect the optimum number of dominant colors by analysing the HMR-ratio behavior, like detecting relative minimums or thresholding its variation, which is left for future work, as it is out of the scope of this work.

In order to evaluate the performance of the methods in the HMR, we propose to calculate the Sum Square Error of quantization (SSE) in the HMR, divided by the total SSE in the image, that we will call the SSE-ratio. Graphically this can be appreciated

in Fig. 7.8 (top four graphs corresponding to the four example images). The abscissas in the graphs show several numbers of dominant colors, from 1 to 5, when dealing with generation of 8-color palettes. The different algorithms (FSCL, FCM, NG, K-MEANS, SOM and MSCL) were simulated 50 times to show the averaged SSE-ratio. As it can be seen, MSCL always presents the smallest SSE-ratio, for all the images and different number of dominant colors. That means that MSCL with dominant-color avoidance is able to maintain a reduced amount of error in the HMR, while the others methods tend to concentrate their SSE reduction in the rest of the image.

7.2.4 CQ Focused in salient colors

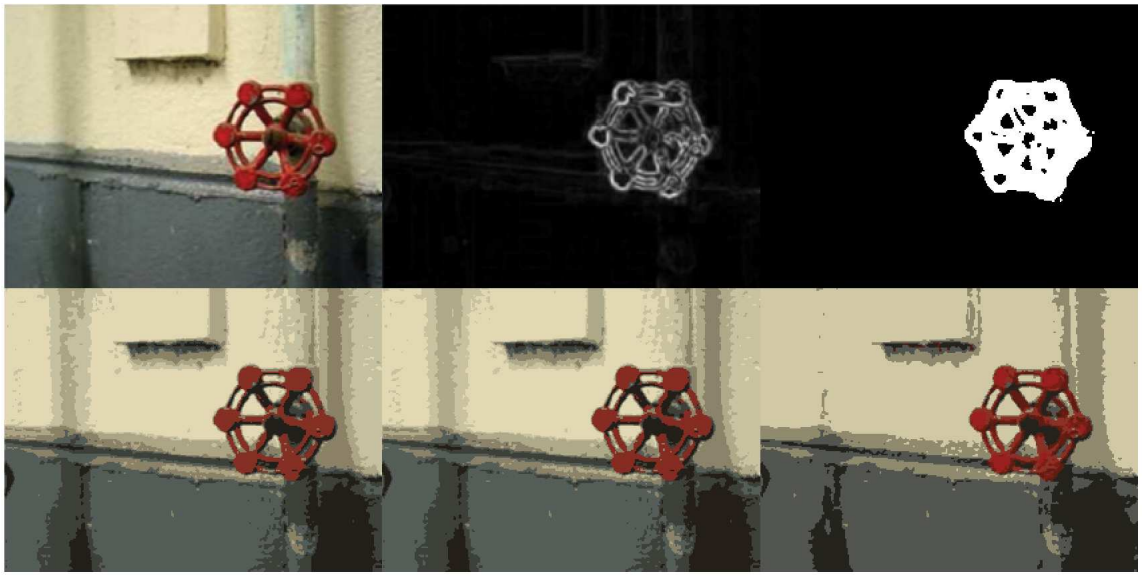


Figure 7.6: Saliency example. *Top row, from left to right*: Original image, saliency map (clearer values for high saliency), the mask binary image used for MSE measurement and (*bottom row, from left to right*) the reconstructed image with an 8-colors palette from: SOM, FS-SOM and MSCL focused on the saliency.

The aim of salient feature detection is to find distinctive local events in images. Some works ([77]) exploit the possibility of color distinctiveness in salient detection. This example shows the MSCL algorithm generating a color palette focused on those salient regions. To achieve that, the chosen magnitude function is the mean computational global saliency (defined as in [77]). The magnitude is normalized by the maximum, and varies from one to values near zero in zones with low saliency (see image in Fig. 7.6 in the middle of the top row). We used 8 colors with decreasing learning rates between 0.7 and 0.01 for every

algorithm.

Figure 7.6 shows an example. The first two algorithms (SOM, FS-SOM) only obtain a red color and present higher MSE values (SOM: 103.21 and FS-SOM: 103.07) in those pixels belonging to the white mask region of saliency (right top image of Fig. 7.6). However, using the global saliency (middle top image of Fig. 7.6) as the magnitude for MSCL, the resulting image shows three red variants and the MSE error is lower (87.5). A drawback is that other colors are under-represented, what means a minor problem if we want to detail the salient regions of the image.

7.2.5 Image binarization

Binarization of a text grey-scale image is the process of assigning each pixel of a text image depending of its grey-scale value to one of two classes, one corresponding to the text and the other one to the background. First row of Fig. 7.7 shows the image of a badly illuminated document (image a), and the results of applying classical binarization algorithms: Otsu method (b), filtering of original image with Laplacian operator (c) and its binarization with Otsu (d). Otsu Method definitely fails to get an adequate binarization because of the dark grey values in the right margin of the paper. Filtering with Laplacian operator provides a better result, because it is an edge extraction mask. However, this method does not fill the letters.

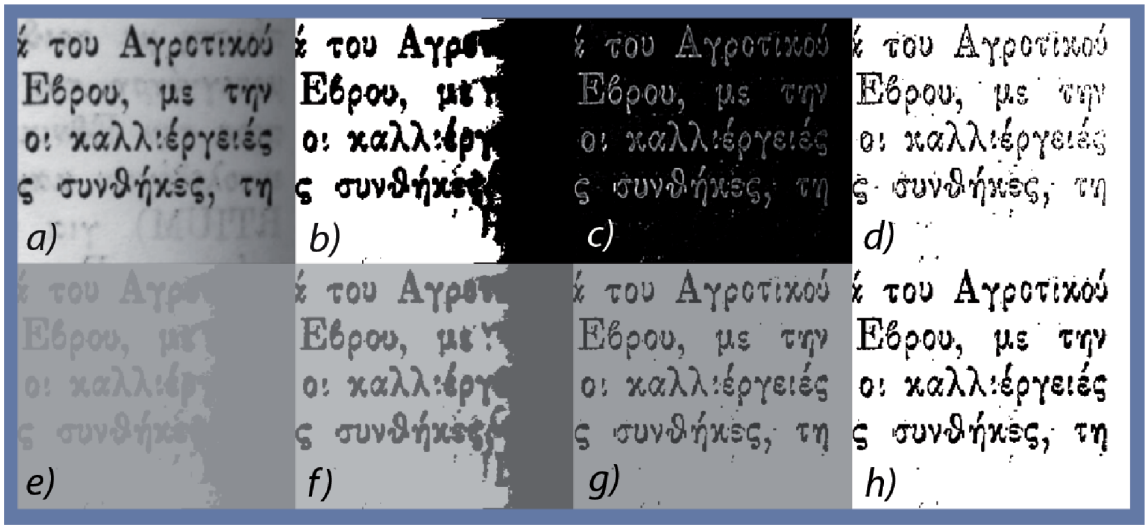


Figure 7.7: Binarization example: in *top row* (a) original image, (b) Otsu method, (c) filtering with Laplacian operator, and (d) its binarization with Otsu; in *bottom row* (e) SOM, (f) MSCL in homogeneous grey quantization, (g) MSCL with two features, and (h) Otsu binarization of (g).

Competitive learning can be used for this application by training 2 units to represent two levels of gray-scale, which should correspond to the background and foreground classes. Second row of Fig. 7.7 shows the results with: (e) SOM, (f) MSCL in homogeneous grey quantization, (g) MSCL with two features (explained below), and (h) Otsu binarization of last example. The MSCL in (f) with only two neurons is equivalent to the Otsu Method. The reason is that the mean quantization error for each unit is proportional to the standard deviation of a data class when using as mean of the data the unit weights that represents the class.

The quantization result can be improved by using as input a combination of the gray-level values and the result of Laplace filtering. Therefore data samples will be two dimensional vectors combining the values of both features. Then if we apply MSCL using homogeneous quantization to this combined dataset we will get the two-level image (g) in Fig. 7.7 (the same image with binarized pixel intensity can be seen in next image (h)). This result is better than those achieved by other classical methods.

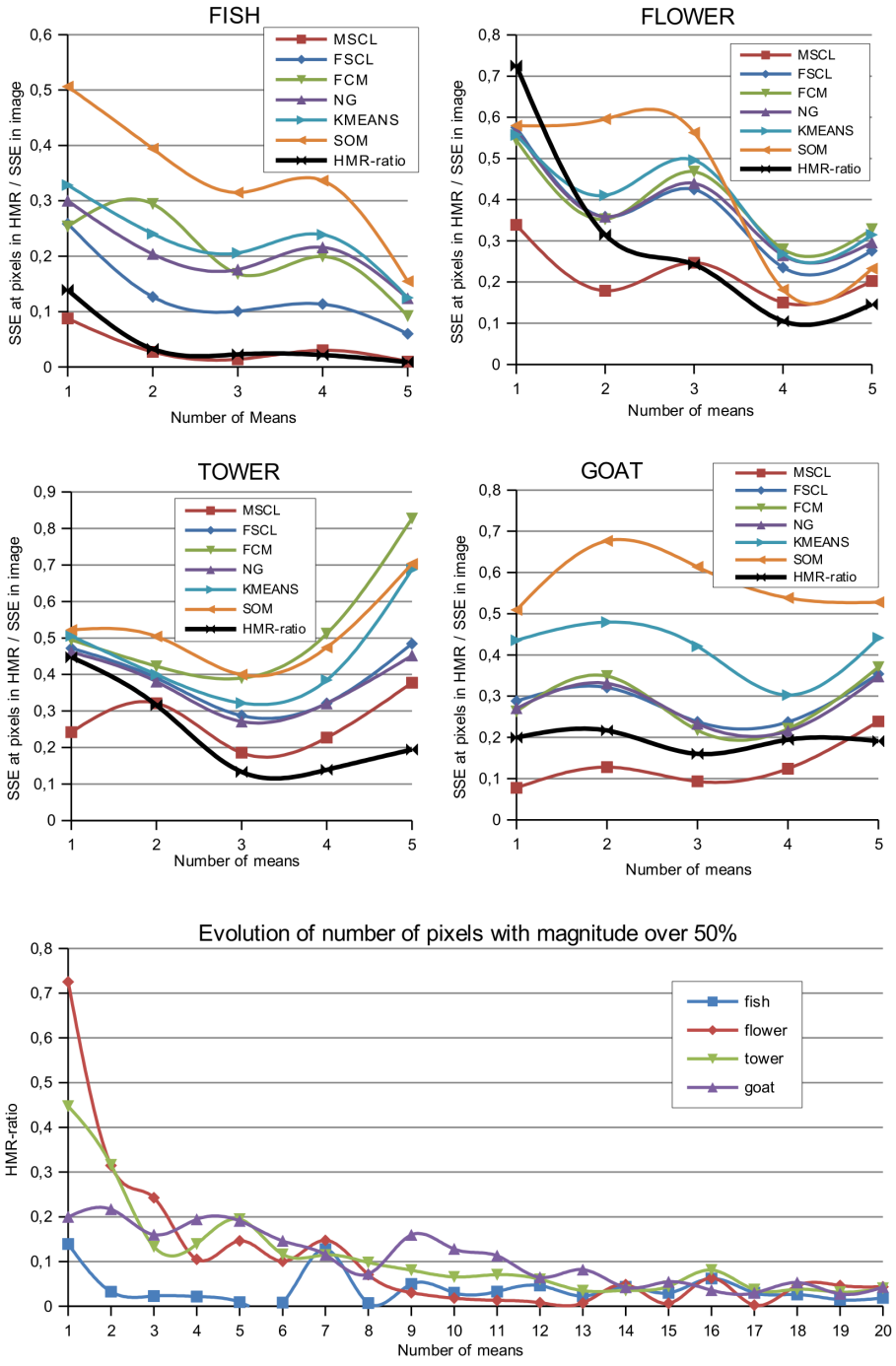


Figure 7.8: The top four graphs correspond to each example image, when dealing with generation of 8-color palettes. The averaged Sum Square Error in the High Magnitude Region (HMR), divided by the total SSE in the image (SSE-ratio) is represented for the different algorithms (FSCL, FCM, NG, K-MEANS, SOM and MSCL). The abscissas in the graphs show several numbers of dominant colors, from 1 to 5. MSCL always presents a smaller SSE-ratio, for all the images and different number of dominant colors. The bottom graph represents the evolution of the averaged HMR-ratios (number of pixels in HMR divided by total number of pixels) when using from 1 to 20 dominant colors.

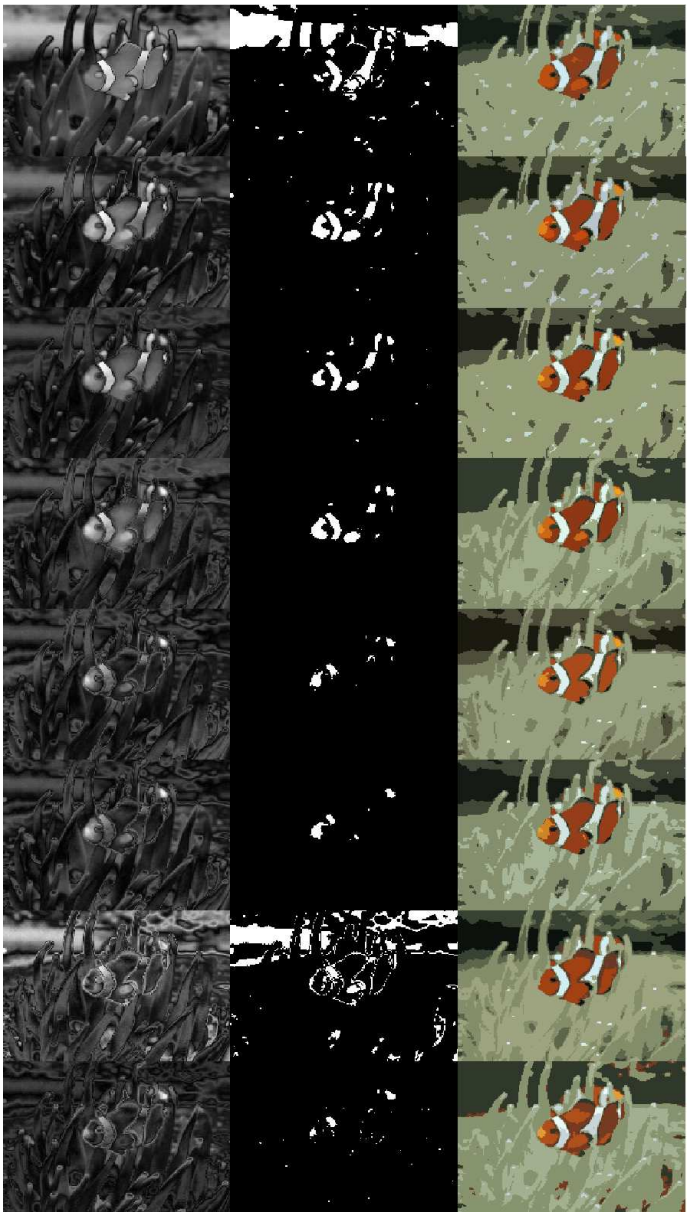


Figure 7.9: Results of CQ of the Fish example in a 8 color palette, avoiding different number of dominant colors: (from top to bottom) with 1 to 8 dominant colors. (*In columns*): magnitude map, pixels with magnitude value over 50% of the maximum (High Magnitude Region), and MSCL reconstruction for the corresponding number of dominant colors.

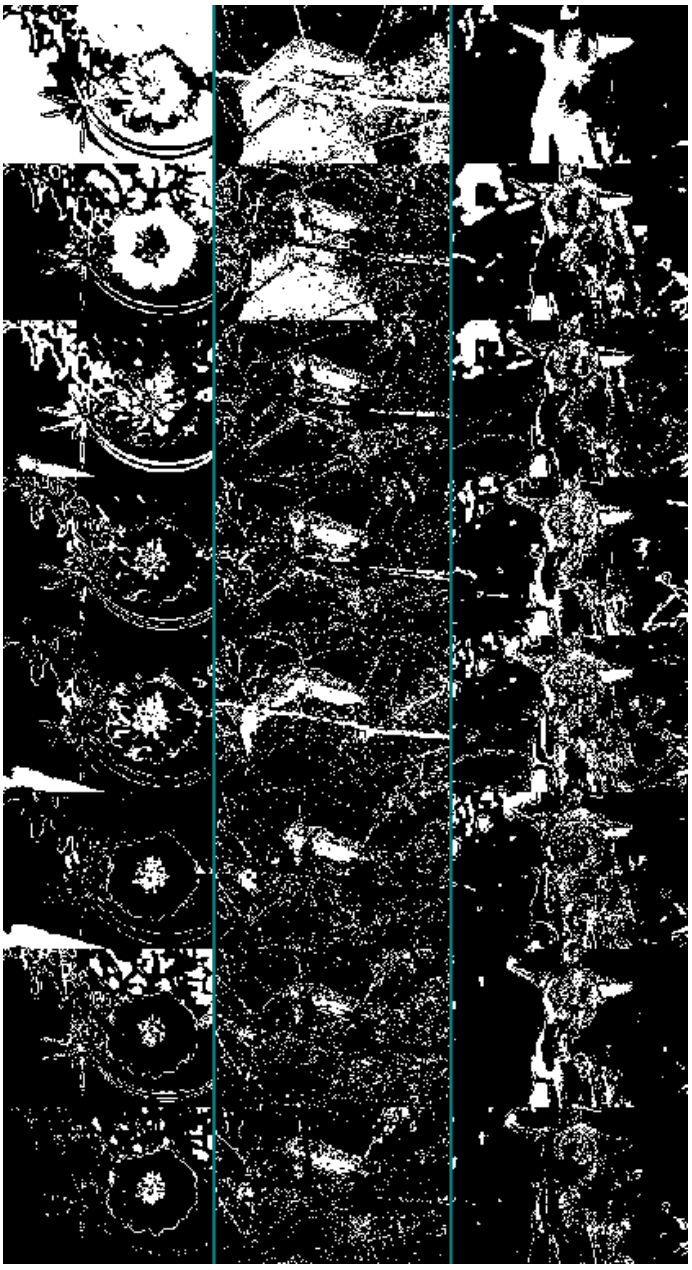


Figure 7.10: High Magnitude Regions for different number of dominant colors. Images in rows correspond, from top to bottom, with 1 to 8 dominant colors. Images in the left column show the flower example, the column in the middle the tower and the right column the goat image.

Chapter 8

MSIC: Magnitude Sensitive Image Compression

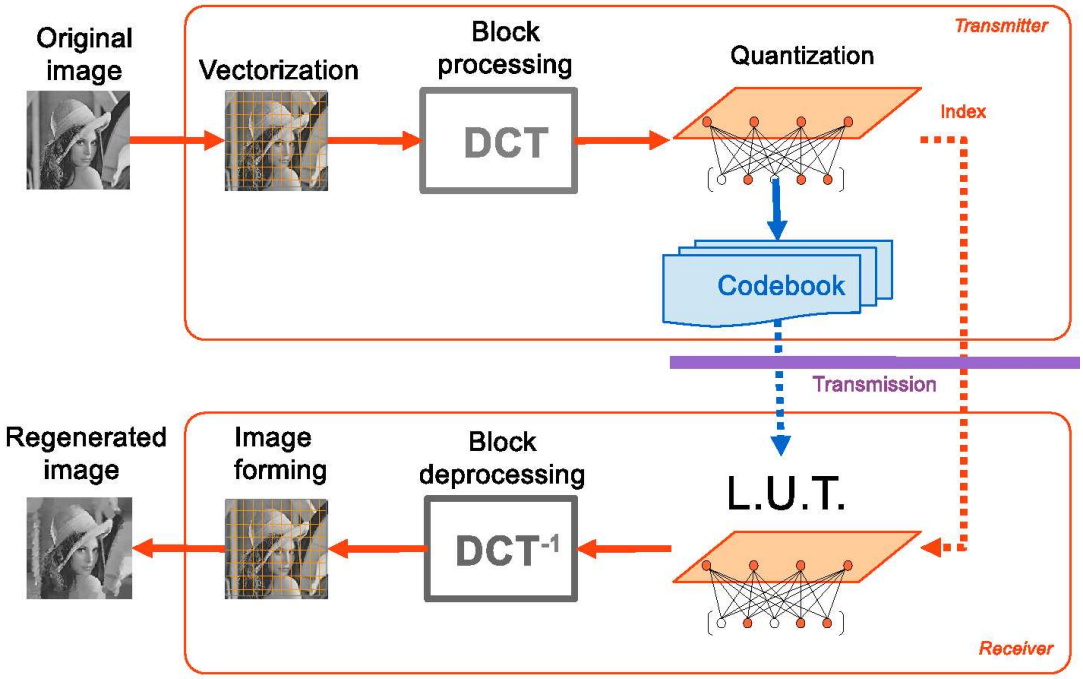
8.1 Introduction

In the human vision system the attention is attracted to visually salient stimuli and therefore only scene regions sufficiently different from their surroundings are processed in detail. This provides the necessary motivation to devise a novel image compression method capable of applying distinct compression ratios to different zones of the image according to their saliency.

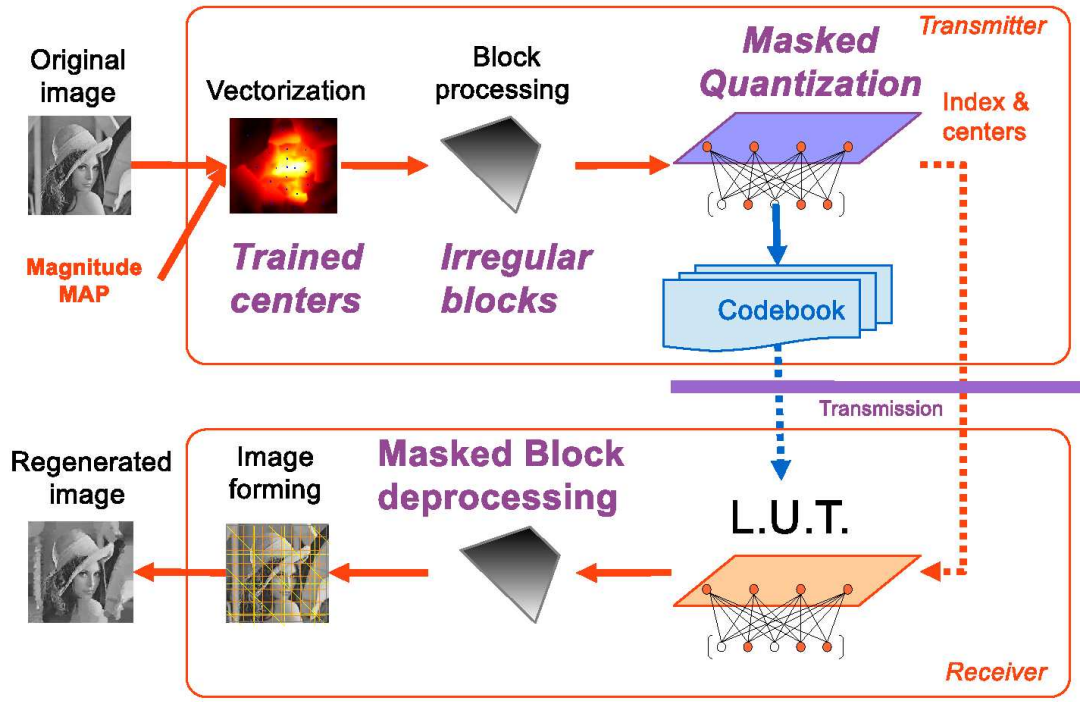
In this chapter we make use of the Magnitude Sensitive Competitive Learning Algorithm (MSCL) to get a sensitive image compression. Using saliency as the magnitude, units tend to model more accurately the salient areas of an image, and therefore the neural network behaviour imitates the human vision system.

In the context of image processing, basic vector quantization consists in dividing the input image into regular blocks of pixels of a pre-defined size, where each block is considered as a *D-dimensional* vector. Each of these input vectors from the original image is replaced by the index of its nearest codeword, so only this index is stored or transmitted through the media. The whole codebook serves as a database known on the reconstruction site. This approach reduces the transmission rate while maintaining a good visual quality. Figure 8.1(a) shows this procedure.

In VQ, compression level depends on two factors, the number of blocks and the level of compression of each block. Both factors are related in an inverse way. Lower number of blocks means that they are higher in size, and therefore higher is the bit depth necessary



(a) Common CL image compression algorithm.



(b) MSIC algorithm.

Figure 8.1: Basic idea of Competitive Learning algorithms in the task of image compression for grayscale images. *Top*: Common CL algorithm. *Bottom*: MSIC algorithm. Differences with other CL algortihms are the use of a MSCL to get block centers (centers are trained weights of MSCL units), the use of irregular blocks and the masked quantization/deprocessing.

to codify each block for a similar quality.

Some authors [41], [5], [26] and [45] have already used some VQ variants, such as Kohonen neural networks [38] for image compression. These algorithms divide the image in block of fixed size and use several tricks to get a smaller codification of each block or to improve the quality of the codification. Laha [41] uses surface fitting of data assigned to each codeword instead of the codeword itself, which improves the visual quality of the results. [5], [26] and [45] apply DCT filtering [2] to each block previous to the quantization step to lower the dimension of the input data. On the other hand, [5] takes advantage of the topological ordering property of the SOM neural network to codify indexes with a few bytes.

In this work blocks may have different size, chosen according to its relevance (which is selected following the image saliency). Blocks located in areas of high image saliency are smaller than those assigned with lower saliency. As bit depth used in the quantization step is the same for all blocks, quantization error increases directly with the block size in areas of low image saliency. Therefore, a lower number of blocks is used to represent the whole image increasing the overall image compression and preserving, at the same time, the quality of most relevant areas.

Another important difference in our approach in relation to the above mentioned methods is that block shapes are, in general, irregular, i.e., neither rectangular nor squared. Therefore, quantization has to take into account samples that may have invalid components. 8.1(b) shows the basic idea of the proposed algorithm applied in grayscale images. It requires to transmit the block centers and index. At the receiver, it is possible to regenerate the shape and mask of each block and locate it into the image, only with its center and magnitude. Then, the block image is regenerated with its index and summed up to form the whole image. In Sect. 8.2 we present the complete algorithm, that is more complex to reduce the amount of data to be transmitted.

The remainder of this chapter is organized as follows. Section 8.2 shows its use to achieve selective image compression focused on the most salient regions of an image with the method that we call Magnitude Sensitive Image Compression (MSIC) applied in grayscale images. Next section extends the algorithm to color images. Finally, a comparative between MSCL and classical JPEG and SOM based VQ algorithms for a high compression ratio task is carried out in Sect. 8.4.

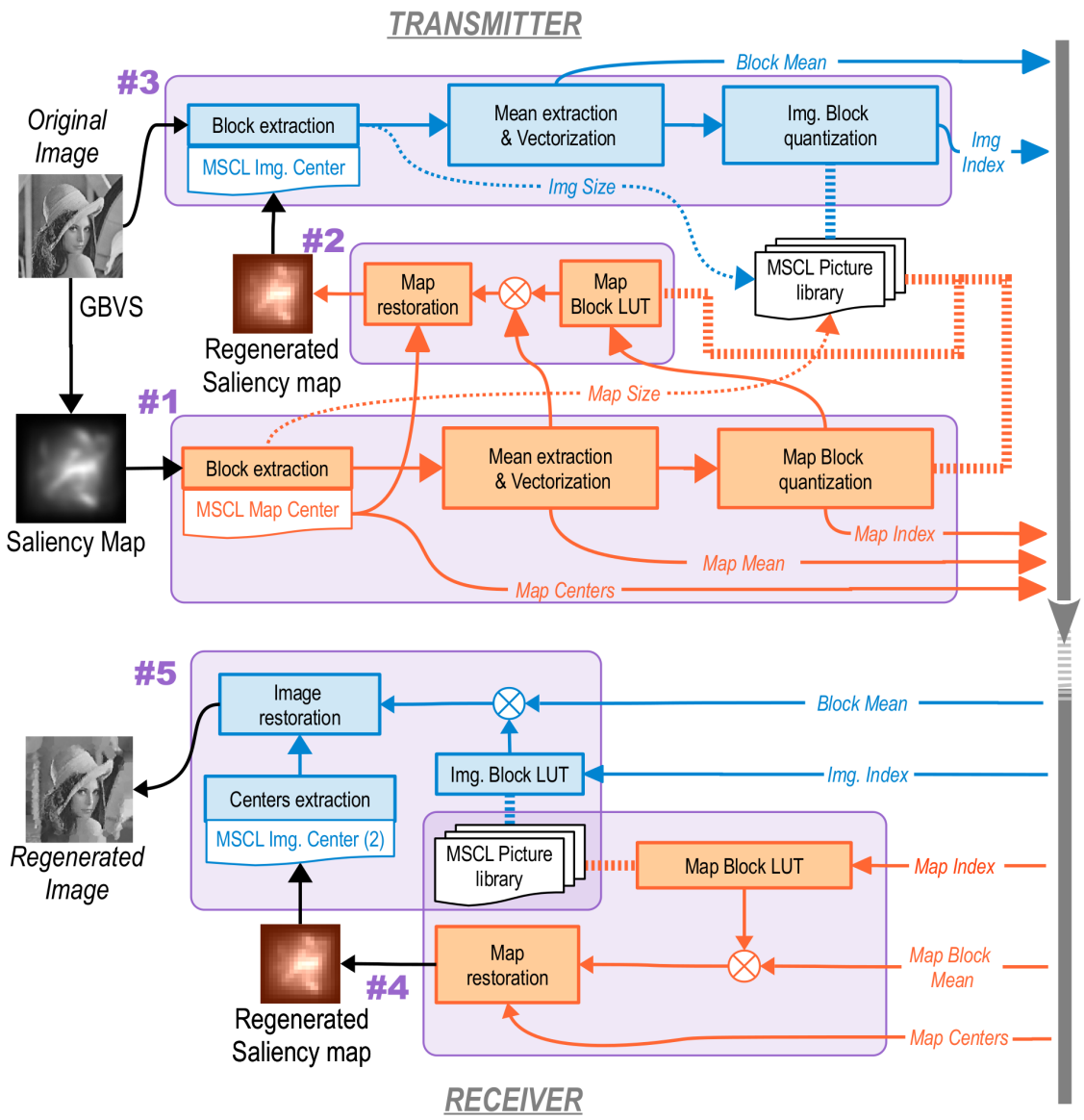


Figure 8.2: Global algorithm for grayscale images. Marked with **#n** the corresponding subsection with the detailed explanation and, also showing the order of processing steps in the transmitter and receiver.

8.2 Magnitude Sensitive Image Compression

Figure 8.2 shows the whole MSIC algorithm applied to grayscale images, where image compression in the transmitter is represented on the top and the image restoration process at the receiver is depicted on the bottom. Image is compressed with different quality according to a selected user magnitude.

In this work we use as magnitude the saliency map, that is an image with the same size as the processed image, obtained by applying a function to the original image. These functions are explained in section 8.4.

The result of the compression is a group of image blocks encoded by indexes. Unlike other image compression methods, our algorithm uses blocks of different sizes, which are located at any position of the image. Therefore, this implies that block centers and sizes has to be sent to the receiver, in addition to the corresponding index. As this approach would mean the transmission of huge quantity of information, we have adopted an alternative solution.

We use the saliency map to train a MSCL network, that we call $MSCL_{MC}$, using as inputs the coordinates (x_1, x_2) of each pixel and the saliency as magnitude. Weights of its units (codewords) of the $MSCL_{MC}$ after training are the block centers $(bc(k), k = 1..M)$. The surrounding pixels assigned to the Voronoi region of each block-center configure the corresponding blocks. The image is so fragmented in so many blocks as units in this network (M), generating smaller blocks in those zones with high saliency and larger blocks in those with lower saliency. In 8.2.2 we will show how to determine the block sizes (and block limits) for each codeword or unit. This process encodes the saliency map with low quality, and both the encoded image and the encoded saliency map are transmitted.

At the receiver, first the saliency map is regenerated, and with it, the image block limits and centers can be calculated. They are used with the image indexes to restore the image.

It is worth noting that it is necessary an additional step at the transmitter. Instead of using directly the saliency map to extract the image blocks, we first decode a saliency map from the encoded map that has to be transmitted. Then we calculate the image centers and limits of image blocks using this Regenerated Saliency Map that will be also regenerated by the receiver.

MSIC algorithm uses several MSCL networks: $MSCL_{MC}$ (map center) to extract map blocks, both with the same number of units, $MSCL_{IC1}$ and $MSCL_{IC2}$ (image center) to extract image blocks at trasnmitter and receiver respectively, and a pool of MSCLs that we call MSCL picture library ($MSCL_{PIC}$) to generate indexes that encode each block pixels, and act as Look-Up-Table to decode the block shapes with these indexes. This

library is calculated using the masked version of MSCL (see chapter 5 of the Thesis) as blocks may have irregular shapes. $MSCL_{MC}$ and $MSCL_{IC1}$ networks are trained online during map and image quantization. Their codewords are used to codify the blocks. However $MSCL_{PICT}$ forms a codebook database that is trained offline. It is known by the transmitter and the receiver as a library of the method. Finally, receiver uses $MSCL_{IC2}$, that becomes identical to $MSCL_{IC1}$ when trained.

Summarizing the MSIC algorithm steps are:

1. Saliency map quantization (at transmitter, creating $MSCL_{MC}$).
2. Saliency map restoration (at transmitter, using $MSCL_{MC}$).
3. Image quantization (at transmitter, using $MSCL_{MC}$ for block processing to create $MSCL_{IC1}$ and codify image content with $MSCL_{PIC}$).
4. Saliency map restoration (at receiver, using $MSCL_{MC}$ received through the transsmision line).
5. Image restoration (at receiver, using $MSCL_{MC}$ for block processing to create $MSCL_{IC2}$ and codify image content with $MSCL_{PIC}$).

Following sections explain the process in detail.

8.2.1 Pictorical library generation

A previous step, before applying the MSIC algorithm, corresponds to generating a pictorical library ($MSCL_{PIC}$), that is known for both, transmitter and receiver. To do it a random set of image blocks of variable sizes is generated. Then it is assigned to a cluster according to its size (we define 7 block clusters), and vectorized. Then, vectorized vector is masked and used as input to train the $MSCL_{PIC}$.

To generate each of the random blocks, we generate the same number of 2D points as the desired number of blocks. Each of these points are the centers of the block, and they are considered as units in a neural network. The block corresponds to the Voronoi region of each of these units. Extraction phase is done as described in the subsection 8.2.4 excepting by the fact that centers are the corresponding coordinates of a pixel selected randomly.

8.2.2 Saliency map quantization

The idea is to consider the saliency map as an image and apply the same compression steps that will be applied to the image. To do it, a MSCL neural network ($MSCL_{MC}$) will be

generated and used for two different purposes. First to define map blocks that are codified with $MSCL_{PIC}$ and transmitted remotely. Then $MSCL_{MC}$ is also used to restore the map at transmitter. This regenerated map image will be used to generate image blocks that are also be encoded with $MSCL_{PIC}$ and sent remotely. Here it is important to note that if the saliency map would be common to several images, emitter and transmitter could know it in advance so this step and the explained in the following section would be unnecessary. In this chapter we will consider that image saliency is calculated directly from each image, and it is necessary to send the saliency map associated to each image.

First step corresponds to the block extraction from the saliency map according to the saliency values. We train the ($MSCL_{MC}$) using the 2D coordinates of each pixel(x) as inputs and the following magnitude function:

$$MF(m,t) = \frac{\sum\limits_{\mathbf{x}\in\mathcal{R}_m} saliency(\mathbf{x}(t))}{|\mathcal{R}_m(t)|} \tag{8.1}$$

where $saliency(\mathbf{x})$ is the pixel saliency of the corresponding sample. Trained unit weights correspond to the coordinates of the unit in the image, and the magnitude value is the mean of the saliency inside its Voronoi region. Once trained, it is possible to find the best matching unit (BMU_{MC}) assigned to every pixel (using magnitude during competition). The block assigned to each unit is the rectangle wrapping its Voronoi region. A block mask of equal size than the block is also provided in order to mark the pixels belonging to that irregular Voronoi region, see Fig. 8.3. We used 40 units for $MSCL_{MC}$ in our experiment. With this small number of units a coarse saliency map is obtained, but it is enough to define areas with high saliency.

To codify each of the blocks by VQ, we first resize the block to a squared shape with side value as the maximum between its horizontal and vertical block sizes. The block and the mask are inserted in the squared image filling with zeros the void rows or columns. After that, both are resized to a vector form. We use mean-removed vectors to have a better quantification. Mean value of saliency in each block of pixels, that we call mean block-saliency (m_b), is sent encoded by 7 bits.

The resulting vector is separated according to its length and dispatched for training or testing to the MSCL picture library ($MSCL_{PIC}(l)$). This pool of codebooks are trained separately only once and become a lookup table in the algorithm. In order to avoid the transmission of the whole codebook, the pool it is known in advance by both the transmitter and the receiver.

Each codebook of the pool, with 256 codewords, is dedicated to a precise input-vector

length. The imposition of the same number of codewords for different block sizes forces that larger blocks present less detail in pictorial content than smaller blocks. We have chosen a limited group of sizes that model several size possibilities (the value of l is the length of the square edge to which we have resized the block): $l = [4, 6, 7, 8, 10, 15, 29]$. Figure 8.4 shows the trained codebook for sizes $l = 4$ and $l = 10$.

This pool of codebooks can be specialized in the type of images considered in the transmission task, or can be generated using an universal library of training images. The images for training are processed following previous described steps, but the magnitude function chosen for these $MSCL_{PIC}(l)$ networks is the hit frequency of each unit m:

$$MF(m,t) = hits(m,t) \tag{8.2}$$

During competition the BMU_{PIC} is calculated using the masked version of MSCL in order to avoid the zero-padding mentioned before. Each time a sample is presented to each neural network of the pool, the corresponding mask is also presented, and only masked weight components are used to compete (see Fig.8.3, Right). Each sample might have different masked components. In this way, only pixels corresponding to the Voronoi region of a block are used to find its BMU_{PIC}.

At the end of this step, the magnitude map has been divided in 40 blocks. It is necessary to send the receiver the following information of each block: Map indexes (BMU_{PICT}) (1 byte), Map mean (7 bits) and Map Centers (2 bytes). It is not necessary to send the size of each block because it is calculated from the block centers. In the codification of the whole image of the saliency map they are used: 40 (blocks) * 31 (bits per block) = 1240 bits.

8.2.3 Map restoration at transmitter

Map representing the saliency of the image, $MSCL_{MC}$, is also restored at transmitter with the information generated at the previous step. This is because the restored map will be used at both transmitter and receiver to define the block centers of the image, to make the results be the same in both sides. Map restoration is accomplished following the previous step in inverse order. First we calculate Voronoi regions assigned to each of the Map Centers by searching for the BMU_{MC} of each pixel in $MSCL_{MC}$. The codewords of this neural network are the Map Centers. Additionally, we calculate block limits and mask wrapping by a rectangle the area corresponding to the Voronoi region of each center.

With $i-th$ index, the new block is converted again into an image block by the look-up table created with $MSCL_{PIC}(l)$. The codeword of the BMU_{PIC} consists of the pictorical content of the block image, but needs to be displaced with the mean block-saliency value

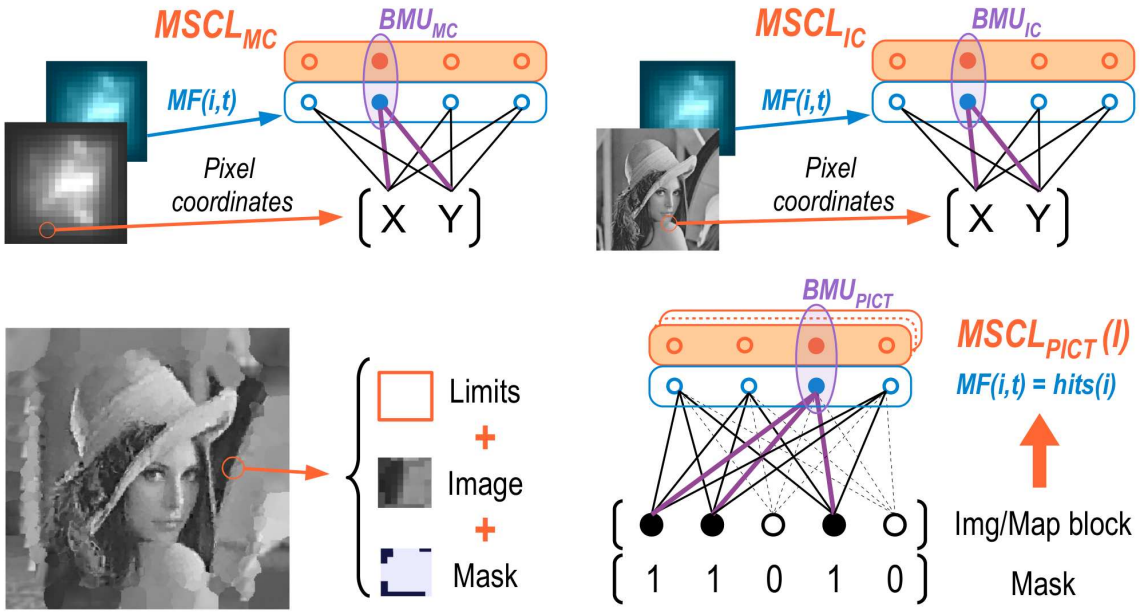
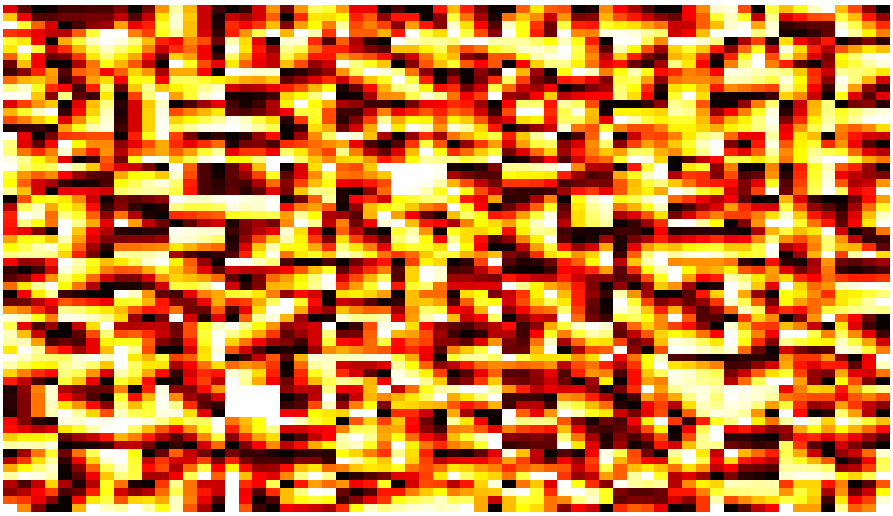


Figure 8.3: Neural networks used in the MSIC algorithm: *Top:* BMU_{MC} and BMU_{IC}. It is important to mention that this last MSCL is used also in receiver (BMU_{IC2}). *Bottom:* Block extraction phase. Each block delivers the block limits, the image and a binary mask. $MSCL_{PICT}(l)$ neural network, where a input sample (vectorized block from the extraction phase) has several masked components.

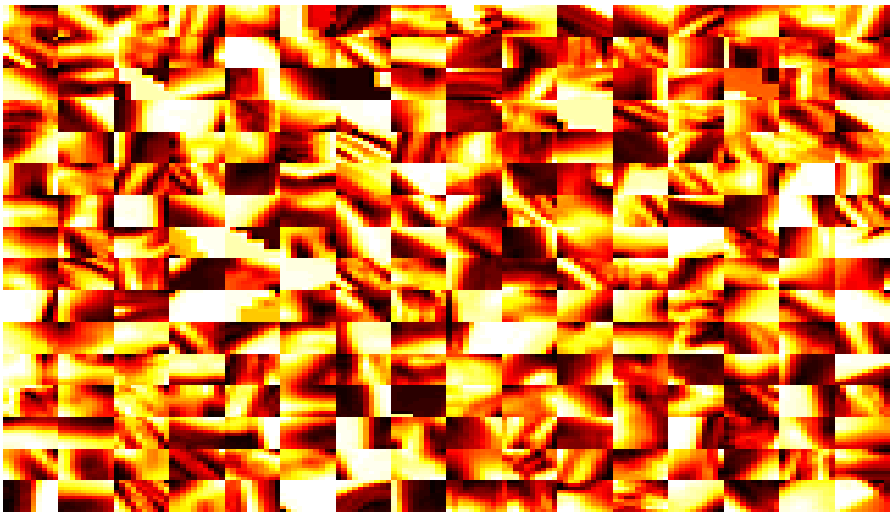
of the corresponding block. After summing the mean, it is masked by the binary mask and added to the regenerated saliency map. Repeating the process for all the blocks we obtain the regenerated saliency map, that will represent the saliency values of pixels for the reconstructed image.

8.2.4 Image quantization

The same process of the previous saliency-map image quantization is followed in this image quantization step. Blocks are extracted training the $MSCL_{IC}$ (with the coordinates at each pixel) to get the image block centers according to the Regenerated Saliency Map at the transmitter. Then the Voronoi regions of each of these centers are calculated. Blocks are extracted and vectorized. After removing the mean, each image block is processed using the masked version of MSCL with the $MSCL_{PIC}(l)$ that corresponds its size, in order to assign it the most similar pictorial content of the library that will be included in the reconstructed image. It is only necessary to send the corresponding block mean and index from the $MSCL_{PIC}(l)$ for each block.



(a) Pictorical codebook of size 4x4



(b) Pictorical codebook of size 10x10

Figure 8.4: Examples of $MSCL_{PICT}$ codebooks. (a) Codebook size $l = 4$. (b) Codebook size $l = 10$. These codebooks and others for different sizes are known, in form of library, by the transmitter and the receiver.

In the codification of the whole image of the saliency map they are used: 830 (approximate number of blocks) * 15 (bits per block, one byte for block index and 7 bits for block mean) = 13690 bits. The number of blocks vary slightly in the tests to get a final number of bytes equivalent to JPEG (including map quantization) for a desired compression quality, as indicated in second column of table 8.1.

8.2.5 Map restoration at receiver

Map restoration at receiver is accomplished following exactly the same process than map restoration at transmitter. To do it, the receiver uses block Map index, mean block-saliency, block-center and the same offline $MSCL_{PIC}(l)$ picture library. As operations are the same and they are applied to the same data, the Regenerated Saliency Map at receiver is exactly the same than the one at the transmitter.

8.2.6 Image restoration

Last step in the whole process is image restoration, using the received means of block-saliency, the pixel indexes and the regenerated saliency map. This step is similar to the previous described Map restoration with small changes.

The main difference is that the image block centers are not available (they have not been transmitted). They are calculated training a new MSCL ($MSCL_{IC2}$), with the coordinates of each pixel, and the magnitude values in the Regenerated Saliency Map (magnitude that was calculated with eq.8.1 at the emitter). This neural network becomes identical to $MSCL_{IC}$. The weights of $MSCL_{IC2}$ are the centers of the image blocks, and their Voronoi regions define the masks and limits.

Once again, image indexes are presented to the look-up table created with $MSCL_{PIC}(l)$ (according to the block size) that returns the block shape. Final image is regenerated by adding means of block-saliency, masking each block and positioning it in the image (adding it to the regenerated image as we had done before with the saliency-map image).

8.3 Extension to color images

Figure 8.5 defines the flowchart to use MSIC in the case of color images. The process is similar to the used in the case of grayscale images, but applied to each of the color components of the image.

First, we calculate the saliency map from the color image. With this saliency map we extract and quantify blocks (as described in Subsect. 8.2.2), blocks which are restored at

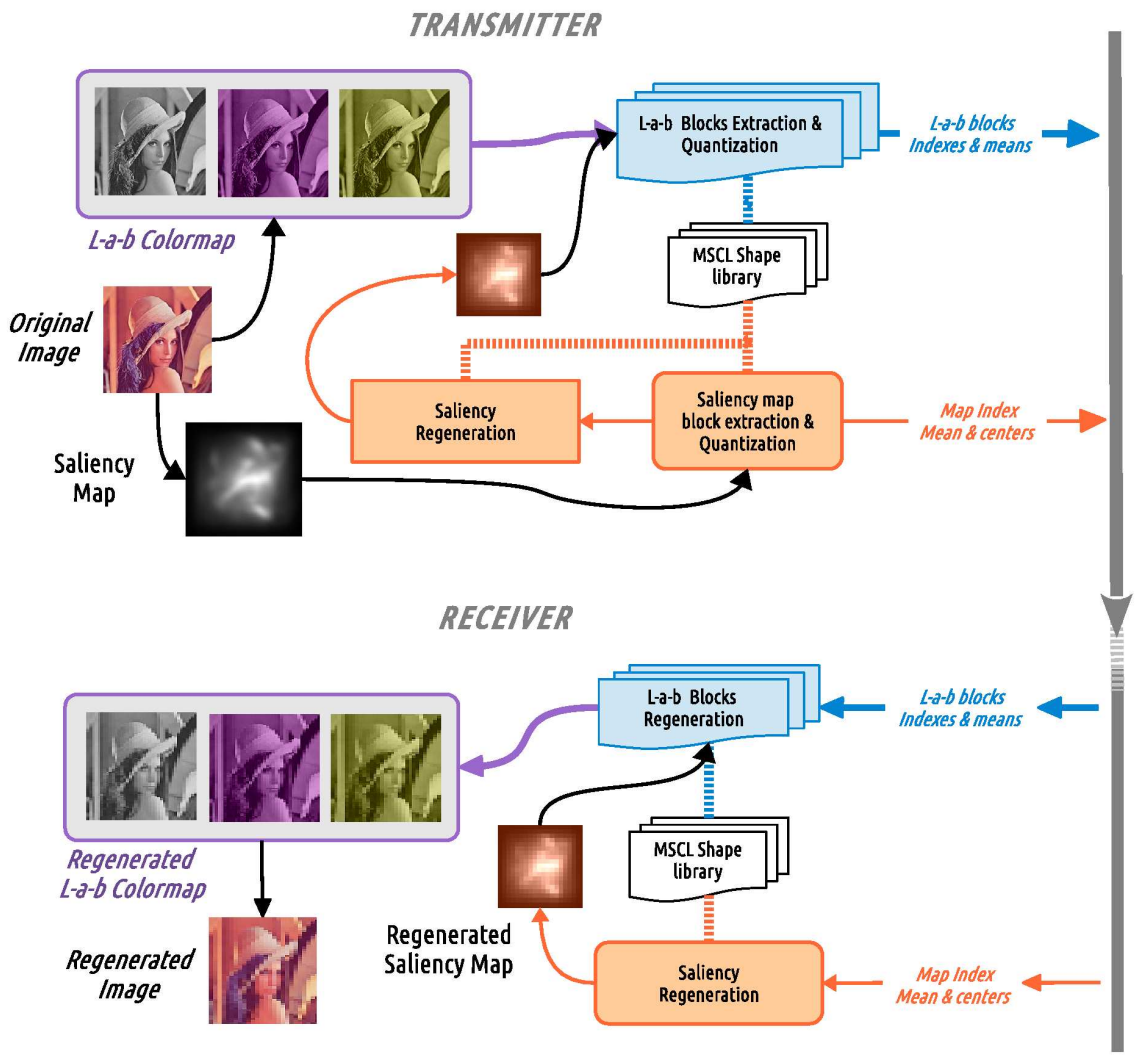


Figure 8.5: Global algorithm for color images. Each color component is processed separately as in the grayscale method. However this process is exemplified in the text with a different magnitude definition for the saliency map, oriented to preserve the detail of the image for certain colors selected by the user.

transmitter as mentioned in 8.2.3. As a result of this step we get the map block-centers, block-means and indexes. Encoding is made with the previously trained $MSCL_{PIC}(l)$ picture library.

Then, original RGB image is transformed to the L-a-b color space. The reason of selecting this color codification is that it has been demonstrated its suitability for interpreting the real world [18].

Now with these L-a-b color components of the image, we follow the process indi-

cated in Subsect. 8.2.4. Each of them will be trained with a MSCL neural network ($MSCL_{IC-L}, MSCL_{IC-a}, MSCL_{IC-b},$) and it will return the block sizes and indexes for each component. The indexes of the blocks are also encoded with $MSCL_{PICT}(l)$.

Once at receiver saliency map is restored (see Subsect. 8.2.5). Then, we follow the image restoration step, applied to each L-a-b component. Its centers are calculated training three MSCL networks ($MSCL_{IC2-L}, MSCL_{IC2-a}, MSCL_{IC2-b},$), with the coordinates of each pixel, and the regenerated saliency map. These neural networks becomes identical to those at the transmitter.

To get the final image, we transform the restored *L-a-b* images to RGB.

8.4 Experimental results

8.4.1 Grayscale images

Simulations were conducted on four 256x256 gray scaled images (65536 bytes), all of them typical in image compression benchmarking tasks.

We applied the MSIC algorithm, with the following MSCL training parameters: 15 cycles and β calculated so the learning factor vary along the training process from 0.9 to 0.05. We used Graph-Based Visual Saliency $GBVS(\mathbf{x})$ ([27]) as the pixel saliency of the corresponding sample. However, it is possible to use other kind of magnitudes to define which areas of the image are compressed more or less deeply.

JPEG was applied with the standard Matlab implementation and a compression Quality of Q = 3 or Q=5 (i.e., with a high compression ratio).

Image	*Q/Bytes*	JPEG(Tot/50%)	SOM(Tot/50%)	MSIC(Tot/50%)
Lena	*Q5/2010*	212.3/340.4	**205.4**/374.0	501.1(18.2)/**211.0(6.1)**
Street	*Q5/2127*	**302.3**/369.0	322.1/465.3	466.2(7.8)/**210.6(4.2)**
Boat	*Q5/1988*	**263.9**/383.7	280.4/486.6	436.4(12.3)/**282.0(5.6)**
Fish	*Q3/2090*	485.7/597.7	**466.3**/904.3	895.8(15.8)/**254.2(9.6)**

Table 8.1: Mean MSE for the whole image as well as for areas with saliency over 50% (grayscale example). Standard deviation is also shown (in brackets).

We also compared with the algorithm described in [45], whose main steps are followed for all the mentioned SOM based algorithms: The original image is divided into small blocks (we select a size of 8x8 to achieve a similar compression ratio to JPEG or MSCL). Then, 2-D DCT is first performed on each block. The DC term is directly send for reconstruction, and the AC terms after low-pass filtering (we only consider 8 AC coefficients) is fed to a

SOM network for training or testing. All experiments were carried out with the following parameters: 256 units, 5 training cycles and the learning factor decreases from 0.9 to 0.05.

The number of bytes used to compress each image was the same for MSCL and JPEG (see Table 8.1) and fixed to 2048 for SOM.

For evaluation purpose, we use the mean squared error (MSE) as an objective measurement for the performance. Table 8.1 shows the resulting mean of the MSE in 10 tests using our algorithm compared to JPEG and SOM applied to 4 test images. We present a second column showing the value of MSE but only calculated in those pixels which saliency is over 50%. Standard deviation is also shown (in brackets).

To obtain the generic pictorial library $MSCL_{PIC}(l)$ we used three additional images different to the images used in testing from [30] with the same training parameters. This number is quite low, but enough to show the good performance of our proposal. However, in a real scenario it would be necessary to use a higher number of images to get a suitable pictorial library. Moreover, we have not used any entropic coding applied to $MSCL_{PIC}$ indexes which would have result in a further compression.

As expected, the MSE value calculated for the whole image area given by JPEG is lower than the one provided by MSIC, because in this algorithm prototypes tend to focus on zones with high saliency while other areas in the image are under-represented.

However, when MSE was calculated taking into account only those pixels with high saliency, MSIC obtained better results than JPEG or SOM. This effect can be clearly appreciated by visual inspection of the images represented in Fig. 8.6. They show how MSIC achieves a higher detail level at image areas of high saliency. In the case of JPEG, it tends to fill up big portions of the image with plain blocks, being unable to obtain a good detail at any part of the image. On the other hand, SOM produces slightly blurred images due to the low frequency filtering.

The new algorithm could also be used in image compression applications with other magnitude functions instead of saliency. Fig. 8.7 shows the compressed results of applying MSIC using different Magnitude Functions to the street image. From left to right, first image is the original one, second image is MSIC using the same Magnitude Functions that the one used in eq.8.1. The Magnitude function in third image corresponds to 8.1, but using $1 - GBVS(x)$ instead of the pixel saliency. The fourth image uses the value of the vertical coordinate (normalized to one) and finally the fifth one uses the value of the vertical coordinate (normalized to one) minus one. It can be clearly seen that depending on the defined Magnitude Function, certain areas are compressed in higher quality than the rest of the image (foreground, background, top or bottom of the image).

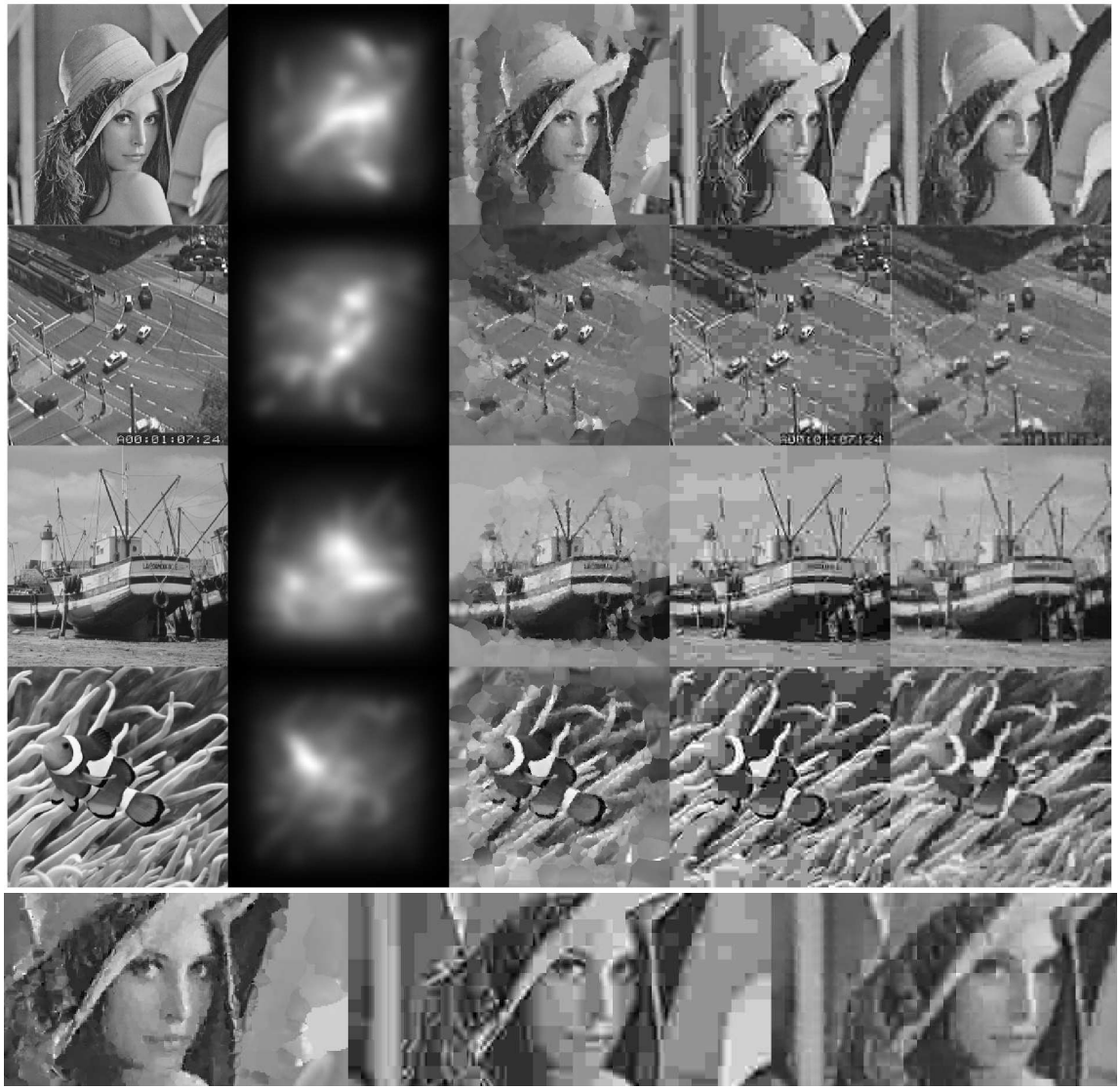


Figure 8.6: *Top in columns*: Original image, saliency map, MSIC, JPEG and SOM compression for the test images. *Bottom*: Lena detail in the three methods. It can be clearly seen that the Lena face, compressed with MSIC shows a more natural view (altmost like painted with Pointillism technique) than the other methods that have square block borders.

Figure 8.7: Original 'Street' image and the compressed images using MSIC with four different Magnitude Functions.

This toy example was only presented to show the possibilities of achieving selective compression in different areas of the image just by varying the Magnitude Function.

MSIC algorithm is much more slower than JPEG. In a serial execution on single core computer, JPEG processing takes only 0.11% of the total processing time used by MSIC (that in our tests it take 6,8 seconds for compressing each of the grayscale test images). Most of the time (91.6%) is spent on block extraction (34% of which is used in extracting blocks from the saliency map and 66% in extracting blocks from the image). Block encoding and decoding takes 6.7% of the time, and 1.7% the rest of the algorithm.

However processing time may be reduced using parallel processing and compiled libraries (now simulated in Matlab). The slowest task is finding the best matching unit to define the Voronoi region to extract a block, and in the encoding-decoding task. BMU finding represents the 68% of the block extraction time, and the 51% of the encoding-decoding time. It is a slow process because in our sequential implementation we must, for each sample, calculate the distances from it to each of the units. In a parallel implementation, this processing could be applied simultaneously for all units. Then, using for instance 1000 units, block extraction time could be only 29.3% of initial total time. Using similar approach for encoding-decoding the final processing time can be reduced to be 2.3 seconds (34,3% of the original processing time).

8.4.2 Color images

In the color experiments, it is applied the method explained in section 8.3, with the same parameters used in the grayscale case.

Now we use a different saliency definition focused in those image zones with colors selected by the user. This type of compression, preserving with more detail image zones with certain color selection, may have several applications. For instance, in medical images, the specialist may define the colors of those areas that has to be well preserved. Other

application is for video transmission limited by narrow bandwidths, as in underwater image transmission. In that case, it is possible to work with a highly compressed global image, and if the user wants a higher definition in areas of a specific color, MSIC could get to a better definition of those areas, obviously degrading others to keep the limited bandwidth.

We propose the next process to calculate the saliency map from the color values of each pixel, we first calculate the saliency map of each color in the set of colors. The saliency map of a selected color is obtained by binarizing the image, based on thresholding the distance of the pixel color and the selected color. Then we apply a border detection algorithm to get the edges of the image zones painted in that color. This edge detection step ensures that the saliency will represent more clearly those zones of the image with borders with the selected colors.

The saliency map of the image is obtained as the maximum of the filtered edge images for all the set of colors. Using this value of magnitude, we get more units in the interesting regions on which colors are similar to the defined set.

JPEG was implemented using Matlab and different compression qualities.

Experiments use the test images depicted in the first column of Fig. 8.8. The second column shows the resulting saliency maps for these images. To keep the fish details in the first image, orange and white colors are used as color set. The flower image uses dark and clear pink, the boat image uses only brown and the parachute image uses pink and black from the parachutist.

Image	*Q/Bytes*	JPEG(Tot/50%)	MSIC(Tot/50%)
Fish	*Q3/1702*	**1328**/2695	2193(20.7)/**1789(40.3)**
Flower	*Q5/1722*	**862**/1299	3540(227.1) /**1167(49.4)**
Boat	*Q6/1720*	**1303**/1570	2366(87.4)/**1190(25.3)**
Sky	*Q5/1706*	967/2312	**240(58.2)** /**468(19.7)**

Table 8.2: Mean MSE for the whole image as well as for areas with saliency over 50% (color example). Standard deviation is also shown (in brackets).

Table 8.2 shows the resulting mean of the MSE in 10 tests using MSCL compared to JPEG. It is also shown for each algorithm, the MSE value calculated in those pixels with saliency over 50%. Standard deviation is shown in brackets, and number of bytes and image quality are also displayed.

As expected, the MSE value calculated for the whole image area is lower using JPEG than the one provided by MSIC. However, when MSE was calculated taking into account only those pixels exhibiting a high saliency, MSIC obtained the best results.

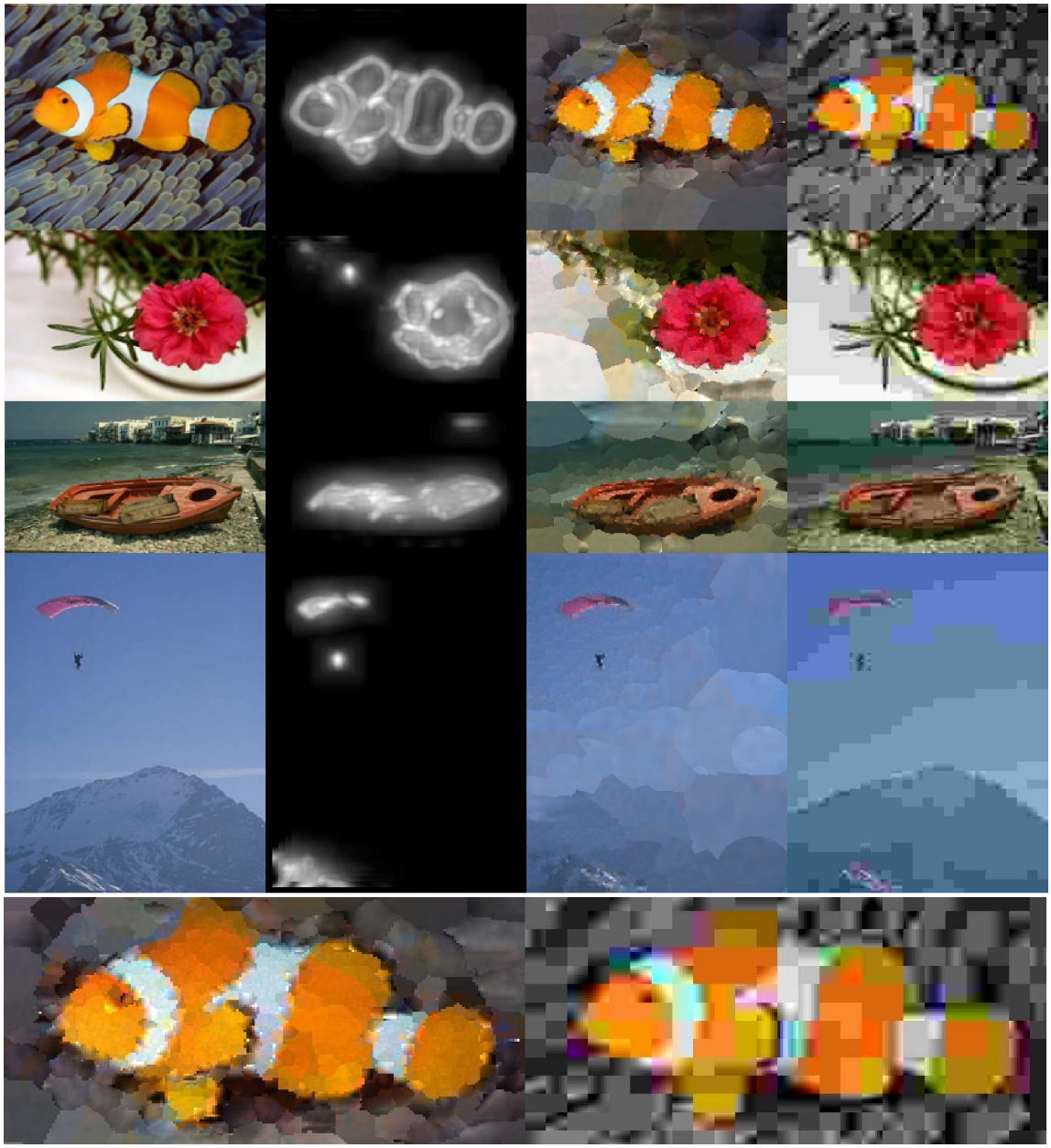


Figure 8.8: *Top in columns*: Original color image, saliency map generated for a one or two-color selection (fish with orange and white; flower with dark and clear pink; boat with brown; parachute with pink and black), MSIC and JPEG compression for the test images. *Bottom*: Fish image detail in both compression methods.

Chapter 9

Surface modelling

9.1 Introduction

The problem of reconstructing the surface of objects from a point cloud is quite common in many areas such as CAD design, cartography, virtual reality and medicine, where input devices can obtain 3D coordinates, but without connectivity information.

Well known techniques propose solutions to the surface reconstruction problem from a geometric point of view. These algorithms require long processing time for the input point cloud because they exploit the adjacency relationship of the data points [29] [4]. For this reason some authors have proposed the use of Competitive Learning algorithms to cope with this problem, specially those that include an intrinsic definition of neighbourhood such as SOM [85] and its derivations [68], [53] and [35].

However these techniques are only capable of centering vertex (units) in the surface according to the 3D-point density, while in many applications it is interesting to detail zones with other properties (i.e., high curvature), and in general these methods require to smooth and reconstruct the resulting mesh structure.

Therefore, we propose to take advantage of the capability of the previously explained Magnitude Sensitive Competitive Neural Networks of distributing units according to a defined magnitude to achieve a mesh surface with high detail in its curved regions, usually pertaining to edges of the surface. This edge detection is interesting for modelling mechanical pieces [74] or modelling 3D environments and robot localization in Simultaneous Localization And Mapping (SLAM) tasks [55].

In this chapter we use MSCL and MS-SOM in two tasks related to 3D surface modelling. First MSCL is used to generate a good 3D representation from real point clouds

captured by a laser scanner. This dataset consists on a large amount of raw data samples with the (x, y, z) coordinates of each point, with no additional information about the surface. Our goal will be to get a surface mesh that model the whole dataset, with unit centroids centered in zones with high curvature. However, as the MSCL algorithm does not include any intrinsic definition of neighbourhood, unlike other algorithms like SOM, we use a neighbourhood definition proposed in section *Algorithm analysis* of chapter 3. Tests were made with three typical point clouds sets, and results are compared with several competitive learning algorithms.

In last section of this chapter we use MS-SOM to perform surface modelling, but from information from 3D depth/range images. This kind of images, have pixel values which correspond to the distance, to points in a scene from a specific point, normally associated with some type of sensor device. Dataset is formed by 3D vectors with the (X,Y) coordinates and the pixel depth. The difference to point clouds is that there is topological information on each pixel (it is known which are the neighboring pixels of a given one). Test were made only with an 3D depth image, and MS-SOM was compared to SOM, just to show its advantages over it. Here, units were impelled to represent with higher detail zones with relevant change on depth in the image.

9.2 Point clouds example

We compare MSCL with other VQ algorithms using two well-known cloud datasets: the first dataset consists of 41255 points from one of the scan views of the Stanford Bunny Model (downloaded from [73]). The second dataset corresponds to the scan of a mechanical piece, the Fandisk. This dataset is formed by 11984 points and was downloaded from [25]. The 3-D coordinates of every point in these databases is used as input data for the competitive models.

9.2.1 Curvature codebook and curvature map

First of all, it is necessary to evaluate the curvature map, that is the value of the curvature in each portion of the input space calculated from the cloud datasets. This value, applied to each point will be used as a magnitude to get a MSCL centered on curvature. But first, we apply MSCL using as target function the mean Q-error, as it was calculated in equation 3.36, to obtain that we call the curvature codebook. This codebook, with prototypes uniformly distributed along the scanned surface, is used to estimate the curvature map that will be handled as a lookup-table for the magnitude curvature.

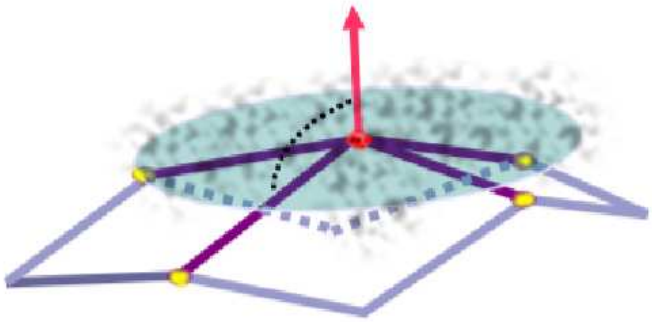


Figure 9.1: Definition of curvature. The cyan ellipsoid is the Voronoi region of unit marked in red. In yellow they are shown the neighbouring units. Curvature at one unit (ie. the red unit) is defined as the average of the projection of each of the vectors between the neighbours and the unit, over the third principal component calculated at the unit's Voronoi region. Red arrow represents this third principal component of the red unit.

In order to estimate this curvature map, we have used the definition shown in figure 9.1, i.e. we consider that the Voronoi data-subset of each unit in the curvature codebook spans a little piece of the surface. This Voronoi-plane is almost flat and centred at the unit prototype. Principal component analysis of these Voronoi data subsets, should generate the first and the second principal components contained in these plane-pieces, while the third principal components should be perpendicular to these planes. For each unit, the vector difference between the unit prototype and those of the mesh neighbour-units is calculated. These inter-prototype vectors are projected on the third principal component previously mentioned and averaged over all the neighbours of the unit to generate its "mean curvature".

When the surface is flat, the units and their closest neighbours will present a value near zero in their mean curvature. However, if the area surrounding a unit is curved, this magnitude will take high values. To maintain this mean curvature realistic, this calculation eliminates from the codebook those units with less than three samples in their Voronoi regions (as three samples only can define a plane and its third PCA component would be null). They usually are units in extreme points of the surface and the estimated curvature would have no sense in these border zones. This pruning process only affects a low number of codewords in the curvature codebook, but does not affect the curvature map estimation, as the data points of eliminated codewords are reassigned to the surviving curvature codebook.

The MSCL for estimating curvature used in the Bunny example included 2013 units, which is approximately the number of points in Bunny dataset divided by 20. For the

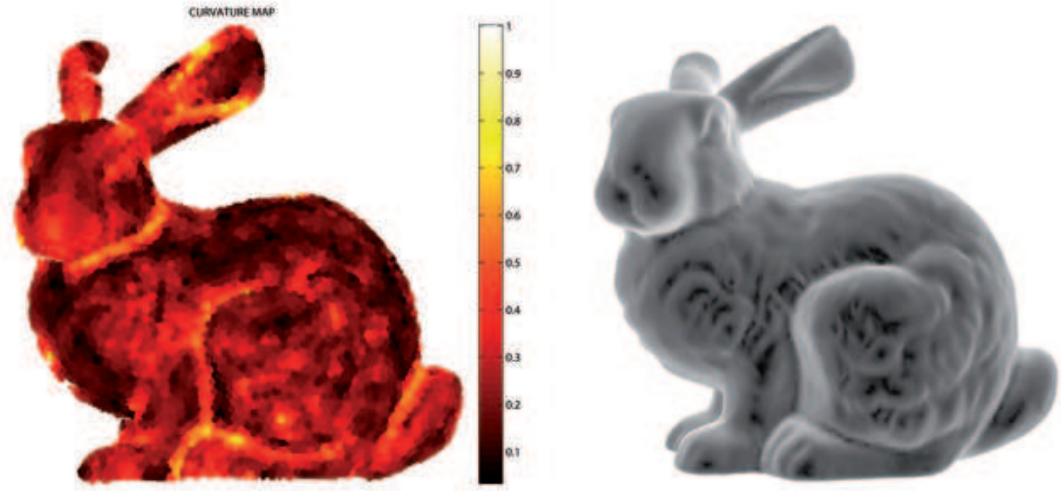


Figure 9.2: In left image, the curvature map for Bunny example obtained with a curvature codebook with 2010 units (after pruning 3 units). In right image, Bunny model visualization from Stanford webpage [73]. It is clear that the curvature map shows enough detail of the model.

Fandisk example the network used 1198 units, which is approximately the number of points divided by 10. For both surfaces, units were initialized with random samples, networks were trained 15 cycles with a value of β, so that initial and final learning rates are $\alpha_{ini} = 0.9$ and $\alpha_{final} = 0.1$. We call the resulting curvature codebook $\mathbf{w}_q$. As the pruning of "border" units reduces the number of units, the final numbers of units were 2010 and 1198 units in Bunny and Fandisk respectively. Figure 9.2 shows in the left image the curvature codebook $\mathbf{w}_q$ obtained for the Bunny example, with the associated color scale representing the curvature values associated to the codewords, while the right image shows the model of Stanford to compare both.

Once the curvature codebook $\mathbf{w}_q$ is obtained, the curvature map for the dataset is calculated as follows: each data sample is shown to the curvature codebook, and its associated curvature is obtained interpolating between those curvatures of codewords corresponding to its first an second BMU.

9.2.2 MSCL focused in curvature

We train a MSCL network denoted as $MSCL_{curv}$, focused on the curvature map of the cloud points surface. This $MSCL_{curv}$ is compared to MSCL with mean Q-error as magnitude, frequency sensitive competitive learning (FSCL), Fuzzy C-means clustering (FCM), Neural Gas (NG), K-Means and Self-Organizing Maps (SOM), using the same parameters

applied in section 3.4.1, but training along 15 cycles. Number of units for the different surfaces were 2013 for Bunny and 2397 for Fandisk. In the $MSCL_{curv}$, the magnitude function for each sample is:

$$MF(\mathbf{x}) = curvature\,(\mathbf{x}) \tag{9.1}$$

9.2.3 Results

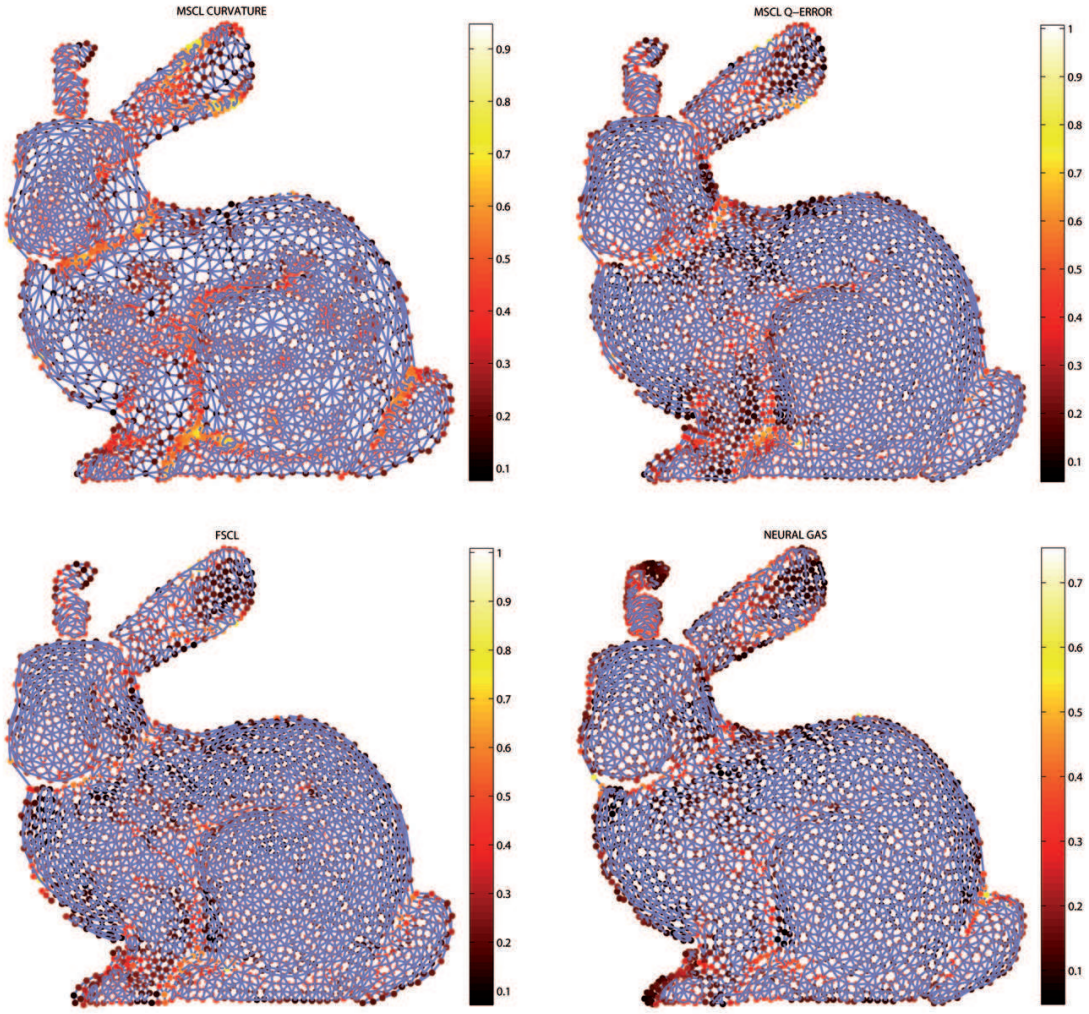


Figure 9.3: Bunny modelling with 2013 units for MSCL with curvature, MSCL with Q-error, FSCL and Neural Gas. Bar color represents curvature values assigned to prototypes. MSCL with curvature concentrates prototypes in the curls and folds of the skin, modelling with high detail the eyes, ears and the joining zones of limbs and body.

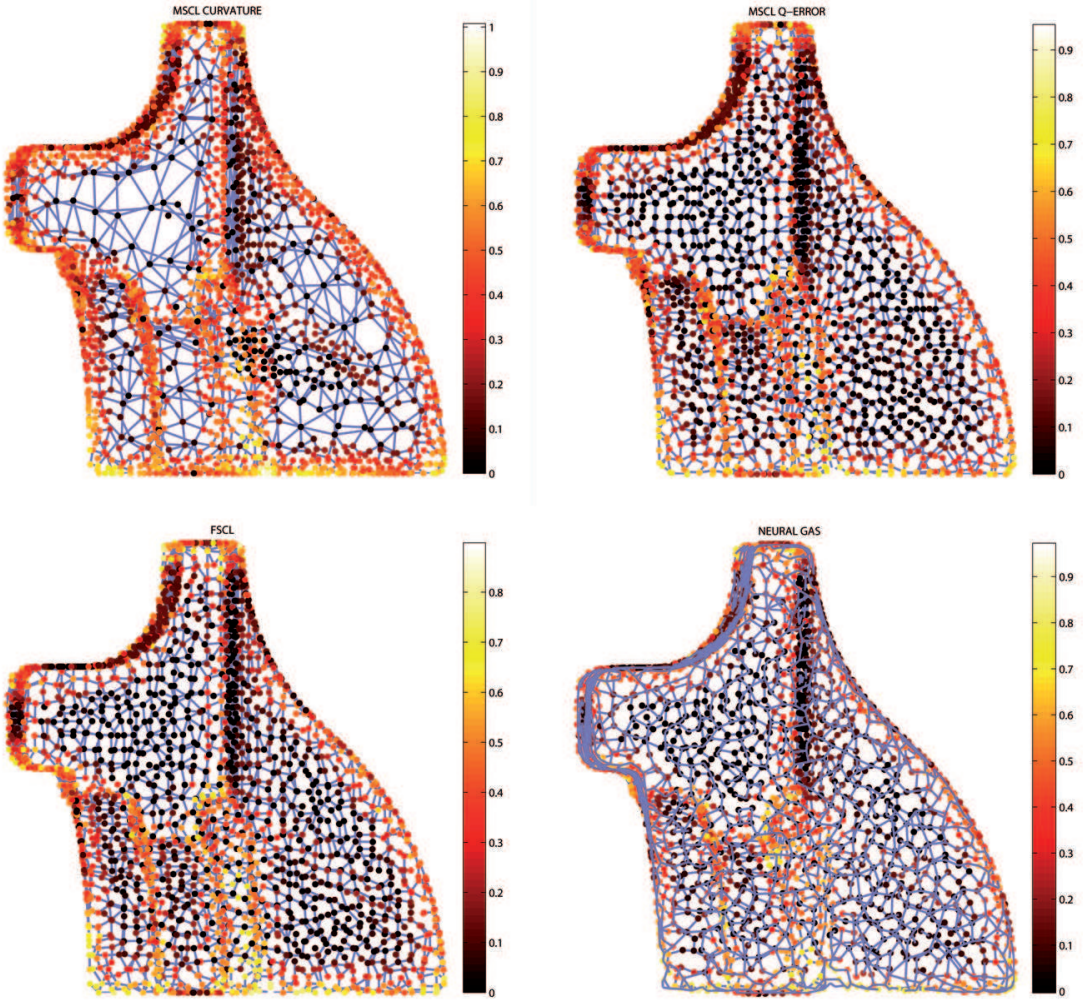


Figure 9.4: Fandisk modelling with 2397 units for MSCL with curvature, MSCL with Q-error, FSCL and Neural Gas. Bar color represents curvature values assigned to prototypes. MSCL with curvature shows more detailed representation in the vertexes and edges of the piece.

In relation to the Bunny dataset, the FCM model cracked, K-means generated many dead units, and SOM result was a too soft representation unable to model the Bunny surface properly. Therefore, Figure 9.3 shows the resulting meshes for the other models: MSCL with curvature, MSCL with Q-error, FSCL and Neural Gas. For the same reasons as mentioned before, figure 9.4 only shows the results for these models in the Fandisk example. It is clear that meshes generated by MSCL with curvature were more efficient in prototype utilization for high curvature zones and edge detections, showing in these zones

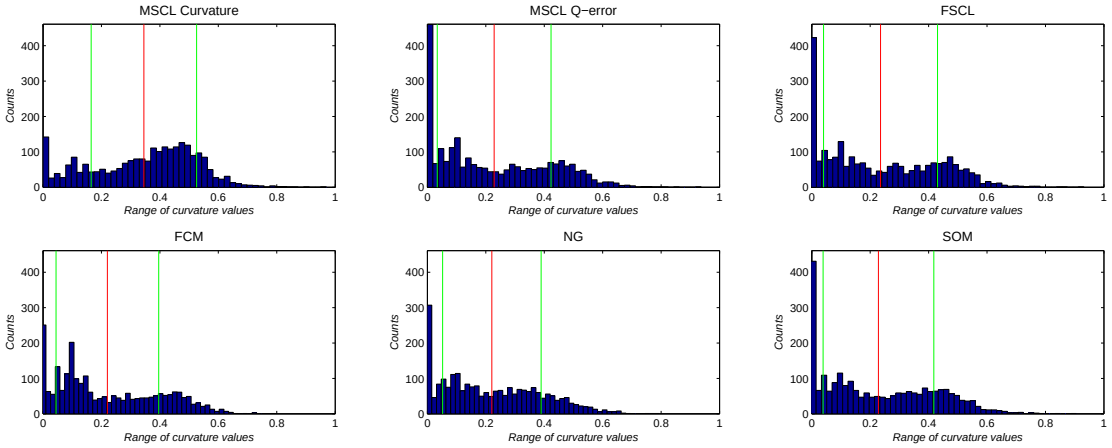


Figure 9.5: Histograms of the curvatures assigned to prototypes in several methods (from left to right and top to bottom: MSCL curvature, MSCL Q-error, FSCL, FCM, NG and SOM) for the Fandisk example. The red vertical line indicates the mean value and the green lines represent the standard deviation range. MSCL with curvature shows the larger number of units in high curvature zones.

a larger number of units that increase their resolution.

As a comparative measure, we represent histograms of curvature values along the different codebooks generated by the models. To assign a curvature value to each unit, we interpolate the curvature map as if its prototype vector was a data sample. Figure 9.5 shows histograms for the different models in Fandisk example. The red vertical line indicates mean value of the curvature distribution for each model. It is worth noting that curvature-based MSCL approach shows the distribution of curvature in its codebook with more concentration in high curvature values taking 0.3451 as mean value, while the rest of models have means between 0.22 and 0.23. The number of units dedicated to planar zones (null curvature bin in the horizontal axe) is clearly the lowest for the curvature-based MSCL approach, while the rest of the models have high number of units in these planar zones.

9.3 3D Depth images example

In 3D computer graphics, a depth map or a 3D depth image is an image that contains information relating to the surface distances of scene objects from a viewpoint, usually represented by a grey level. This property facilitates the computation of the curvature at each region of the image, which is closely related to the problem of discovering the edges

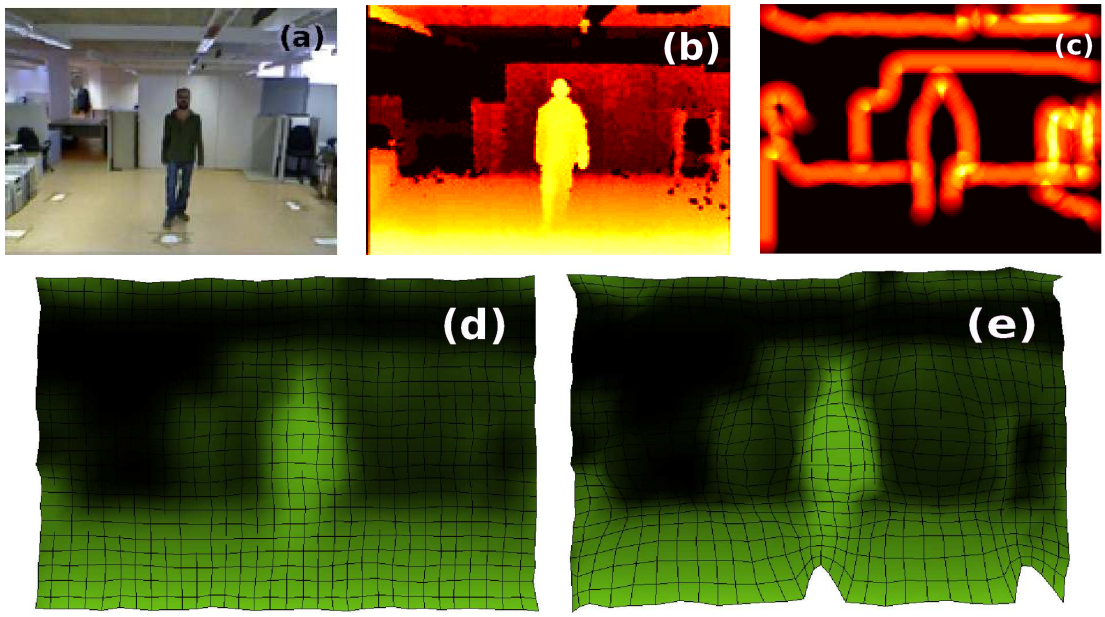


Figure 9.6: *Surface modelling example.*(a) Original image. (b) 3D depth image. (c) Curvature map applying Canny. Zones with higher curvature are also brighter. (d) Final surface models after training dataset with a SOM and (e) a MS-SOM following curvature.

in the grayscale image. An edge mostly corresponds to a change in depth, and therefore it is a region with high curvature.

In this example we compare the performance of SOM and MS-SOM in the task of modelling a 3D surface, given a depth image downloaded from [76]. Processing a 3D image in this way is useful when it is necessary to have a good representation of the 3D space (i.e. in robot indoor navigation to define the 3D occupancy grid mapping).

Data samples are the three dimensional vector formed by the pixel coordinates and the pixel depth: $\mathbf{x}(t) = (x, y, z)$, where z is the normalized distance to the camera. Curvature is calculated applying the Canny filter to the image I. Top of figure 9.6 shows the original image, the depth map I (closer points are brighter) and its associated curvature values obtained with Canny filter. This curvature is used as magnitude vector associated to dataset $\mathcal{X}$.

We trained both SOM and MS-SOM with the same number of units, and similar training parameters, including the same linear codebook initialization. Figure 9.6 (d and e) shows the surface modelled with SOM and MS-SOM respectively. MS-SOM allocates more units than SOM in the zones corresponding to the edges in the 3D depth map, therefore three-dimensional borders are clearly represented, and it is possible to distinguish the

human figure and other details, while in the SOM figure they are mostly confused with the background.

Part IV

CONCLUSIONS

Chapter 10

Conclusions and future work

The subject of this thesis is to advance in dataset modeling via competitive learning where data samples are weighted by a magnitude that does not necessarily correspond with data density.

I proposed a set of *Magnitude Sensitive Competitive Neural Networks* (*MSCNN*) that work like usual competitive learning neural networks in vector quantization tasks, but include a target magnitude function. The effect of this factor is to force units to concentrate in zones of high value of the desired target function, calculated locally from the data or unit parameters. MSCNNs differ from other standard Competitive Learning algorithms that usually generate a discrete approximation to data probability density-function. As a result, *MSCNNs* are more versatile to distribute prototypes following any property or characteristic of the data.

The application examples showed MSCNN capabilities in different applications: Gaussian distribution quantization, data series interpolation, surface modelling from 3D point clouds, color quantization and selective image compression. The comparative results with other competitive methods have validated advantages of MSCNNs in those tasks where the desired codebook distribution does not correspond to the data density distribution.

10.1 Contributions

The major contributions of this Thesis are:

1. I proposed a hard competitive learning algorithm, MSCL, which has the property of distributing centroids in data-distribution zones according to an arbitrary magnitude calculated or obtained locally for each unit (Algorithm is described in [63] and

exhaustively explained in [61]). The algorithm uses two heuristic methods to get this goal:

- First, competition step takes a global and a local phase, where units compete to get minimal product of magnitude by distance.
- Second, learning factor is different for each unit, and depends on the previously accumulated magnitude that this unit have received.

The MSCL Algorithm was proposed in sequential and in batch implementations. Several examples showed the advantages of it over other competitive learning algorithms.

2. I proposed a soft competitive learning algorithm, MS-SOM (inspired in Self-organizing maps), which also has the property of distributing centroids in data-distribution zones according to the magnitude, but keeping track of neigboring relations between units so it produces a low-dimensional (typically two-dimensional), discretized representation of the input space of the training samples, called a map (Algorithm described in [65]).

   Apart from the local competition and the use of specific learning factor for every unit, it uses a neighboring function to modulate learning, and updates all units each time a sample is presented. The algorithm is also introduced in online and batch versions. Experimental comparisons were made with SOM in a simple VQ example and classification tasks.

   The topological representation of stimulus naturally emerges in the biological model of lateral connectivity with excitation/inhibition in the form of Mexican hat. SOM algorithm was developed as an smart simplification of this biological model. MS-SOM introduces a second level of organization of neurons following any magnitude function. This magnitude mechanism could be simplifying other types of biological processes as, for example, a magnitude derived from a chemical diffusion map. This proposition is not supported by experimental biological proofs, but I considered interesting to develop a new method that, preserving the topological behaviour, added other levels of organization with certain biological plausibility.

3. I presented a masked version of MSCL and MS-SOM that enabled the possibility of learning from data samples of different length (idea outlined in [64]). The algorithm is useful when data samples have some invalid components. This situation occurs

some times when data is retrieved from different sources, for instance in statistical data from different countries.

A conventional algorithm fails in learning from this dataset, as it needs some kind of data normalization. I demonstrated in Chapter 5 that a codebook trained with this algorithm results in similar unit allocation than using MSCL or MS-SOM with the unmasked dataset.

4. I presented a generalization of two common algorithms, frequently used in initializing K-means methods: KKZ and K-Means++. The new algorithm, called MS-INIT is the extrapolation of both algorithms to the situation where each sample have an associated magnitude.

   The idea is that MS-INIT also use a 'magnitude' factor during competition to select each of the units from the data samples. This algorithm was compared with other initialization algorithms and I demonstrated that it performs better in initializing a MSCL neural network.

5. I also developed the Magnitude Sensitive Image Compression algorithm, MSIC ([64]). This algorithm, which uses several MSCL networks, was intended to achieve image compression with different compression ratios in different areas of the image, depending on their saliency. The algorithm was developed in two versions, for grayscale images, and for colour images.

   The basic idea of this algorithm was to divide the image in irregular blocks of different size, smaller in zones of high interest (i.e., high magnitude), and bigger otherwise. Then, blocks were encoded by a set of MSCLs, where each MSCL neural network was trained with blocks of different size. As there were more number of blocks in the most interesting areas of the image (areas with higher saliency), these zones receive higher resolution in the compression process.

   The definition of blocks were made by an additional MSCL trained to allocate its units in areas corresponding to the more salient pixels. For training some of the MSCL, I used the 'masked' version of the algorithm.

   Comparison with JPEG and SOM demonstrate the advantages of MSIC in selective image compression.

Additionally I made some minor contributions:

1. I defined two functions to measure the goodness of the quantization achieved by MSCNNs developed along the thesis: WMSE, the weighted version of MSE (Mean

Squared Error), and weighted Entropy. Here, instead of the density of probability, it is used probability of accumulated magnitude at each unit.

2. I also demonstrate the advantage of both MSCL and MS-SOM in a surface modeling task. In this case two real application examples were shown, surface reconstruction from 3D point clouds generated form real laser, and the same task using Depth images captured with TOF cameras. Both algorithms surpassed other typical competitive learning algorithms, because MSCL or MS-SOM were trained to focus on the zones of great curvature, while plain portions of the surface were underrepresented.

3. I presented the use of the new algorithms in a real Color Reduction task. Here, dataset consists on triplets formed by the three coordinates of each sample in the image color space. Training a neural network with a reduced number of codewords, it is possible to get a reduced color palette for the image. The novelty was the use of a MSCL for this training process, with some kind of magnitude associated to each sample. I tested different functions to define this magnitude: pixel saliency, distance to the center of the image, a magnitude that avoids mean color in the image, text image binarization . In these cases, MSCL and MS-SOM performed better than other algorithms.

4. Finally I compared SOM and MS-SOM in classification tasks. In this case, magnitude was selected to force units to focus in zones with higher miss-classification error so the final classification error was lower in MS-SOM than in the SOM algorithm.

10.2 List of publications

1. E. Pelayo, J. D. Buldain, C. Orrite, SO-VAT: Self-Organizing Visual Assessment of cluster Tendency for large data sets, in proceeding of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN (2011) 141-146.

2. E. Pelayo, J. D. Buldain, C. Orrite, Magnitude Sensitive Competitive Learning, in proceeding of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN (2012) 305-310.

3. E. Pelayo, J. D. Buldain, C. Orrite, Focused Image Color Quantization using Magnitude Sensitive Competitive Learning Algorithm, in Proceedings of the 4 th International Joint Conference on Computational Intelligence, IJCCI, (2012) 516-521.

4. E. Pelayo, J. D. Buldain, C. Orrite, Magnitude Sensitive Competitive Learning, Neurocomputing 112 (2013) 4-18, ISSN 0925-2312.

5. E. Pelayo, J. D. Buldain, C. Orrite, Magnitude Sensitive Image Compression, in Proceedings of the 5 th International Joint Conference on Computational Intelligence, IJCCI, (2013) 370-380. (*This paper received the NCTA 2013 - BEST STUDENT PAPER AWARD*).

6. E. Pelayo, J. D. Buldain, C. Orrite, Color Quantization with Magnitude Sensitive Competitive Learning Algorithm, LNCS, Transactions on Computational Collective Intelligence, Special Issue IJCCI 2012. (*Review pending*).

7. E. Pelayo, J. D. Buldain, C. Orrite, Selective image compression using MSIC algorithm, LNCS, Studies in Computational Intelligence (SCI) published by Springer-Verlag, Special Issue IJCCI 2013. (*Review pending*).

8. E. Pelayo, J. D. Buldain, MS-SOM: Magnitude Sensitive Self-Organizing Maps, in Proceedings of the 10th WORKSHOP ON SELF-ORGANIZING MAPS, WSOM (2014) (*Review pending*).

10.3 Future work

Algorithms proposed in this thesis can be improved, and research and future developments can be expected.

Regarding the *methodological aspects* there are some possible competitive learning variants that can be developed, maintaining the idea of magnitude modulation:

1. Regarding MSCL itself, a wider experimentation could be done on some of the algorithm parameters. For instance, using a variable number of winners to compete in the local competition step. In the chapter devoted to the MSCL algorithm, we demonstrated that low values for the number of competing units were the most suitable when a constant value was used. However, it was not investigated the effect of using a variable number of winners, number that could decrease during training.

2. The neural gas (NG) is a simple algorithm for finding optimal data representations based on feature vectors. It is a soft competitive algorithm, as SOM neural network, but NG does not preserve the topological properties of the input space. It has the advantage over SOM of being simpler, and getting better results in some occasions,

specially with high dimensional datasets. Extending the idea of magnitude to Neural Gas as we did with SOM is one of the possible future research lines (this algorithm could be called MS-NG).

3. Another option is using once again soft competition, but with no fixed neighboring relation between units, just winner unit could pull from neigboring ones. These relations could be established between winner an rival units during training (as mentioned in the section related to connections, in the chapter dedicated to MSCL). The resulting algorithm could be an alternative to MS-NG or MS-SOM. The advantages of this algorithm over MSCL would be a faster convergence and the fact that it would avoid 'dead-units'.

4. A growing version of MSCL could be implemented (it could be somehow as an initial step with MS-INIT, and then normal training). The advantage over MSCL or MS-SOM is that it would not be required to know in advance the number of units necessary for quantification.

5. It could be implemented a Recursive/Recurrent MSCL to be applied in temporal series. Then, MSCL would be focused on responding to certain temporal variations.

6. Another methodological improvement is the use of Hierarchical MSCLs. Different MSCLs could be trained simultaneously from the dataset, but achieving better results than if being alone.

7. Finally, it is also possible to use any of the MSNN neural networks to train the center vectors in a Radial Basis Function Neural Network. Therefore, depending on the used magnitude function, it is possible to approximate a function with great detail in certain areas of the space, or achieve better time-series prediction giving higher importance to some of the temporal sequences.

Concerning *the application* of these algorithms, there are several open research lines.

1. A possible future work in color quantization would be the processing of multi-channel images from satellite. In this application, the number of colors is replaced by the number of channels, so the dimension of the data can increase considerably. Using MSCL, with magnitude maps obtained from labelled zones of the images, it is possible to focus the vector quantification towards zones of interest, for example detailing built areas or areas that contains a specific crop.

2. In image compression, future work comprises several research lines such as the use of entropy coding for the information of each compressed image block, filtering each image with DCTs, and its comparison against other compression algorithms. Another point to be analysed is the kind of images used to generate the generic pictorial codebooks used for compression and restoration, as the library of training images can be selected for the chosen task. The test of the algorithm in specific tasks as mentioned in the previous paragraph is a research line that is left for future work too.

3. Surface modeling from 3D point clouds could also be improved using some of the typical methodologies in this type of applications. For instance, it is possible to filter outliers during training (points so far from the main dataset usually are reading errors from the laser scan). On the other hand, it is typical to apply some kind of post processing to the final surface (to prune or create new nodes in some of the areas as border limits, abrupt surface changes, ...).

4. Apart from computer vision, MSCNNs may be applied in other fields. Novelty detection (also known as anomaly detection) is one of these areas. The issue related to novelty detection is that usually are not enough anomalous (or 'new') samples, and training has to be reduced to the known cases.

   Classic competitive learning algorithms provide a solution to this problem, because they are trained with the 'known' dataset, and therefore, this dataset is modeled according to its data density. Then, in the recall phase, each training sample is presented to the neural network. If it is close enough to the neural network, sample is labeled to be 'known', otherwise, it is labeled as 'novel' or 'new'.

   The distinction between 'new' and 'known' samples is improved if units are forced to the data zones with low density. This process may be done using 'magnification' control, as explained in chapter 2, or with MSCL or MS-SOM neural networks. In that case, magnitude corresponding to areas with low density should be greater than its value in low populated regions.

   However, MSCNNs not only can improve novelty detection in this way. The use of competition by magnitude opens the possibility to the use of alternative definitions of 'new' samples. For instance, if samples are ordered sequentially, appearance time could be taken into account so if a sample locates in 'old' regions (that have got no hits recently), it may be considered a 'new' event. Or the novelty may be related to high changes of density. Or directly, a person may label in a supervised way each

sample in the training dataset as 'novel' or 'known'. None of this novelty definitions may be addressed with the classical methods.

5. Clustering is another research field to be explored more exhaustively (I developed a preliminary work in this area in [60]). With the new methods developed in this Thesis, it is possible to use any definition of limit between clusters, distance, distance intracluster versus distance intercluster, or other complex definitions, and force units to model in detail limits among clusters. Then, as in the example of classification with MS-SOM, they may be distinguished in detail limits among clusters.

Conclusiones

El objetivo de esta tesis es el avanzar en modelado de conjuntos de datos por medio de algoritmos de aprendizaje competitivo en los que las muestras son ponderadas por medio de una magnitud que no se corresponde necesariamente con la densidad de los datos.

Se han propuesto un conjunto de MSCNNs que trabajan como redes neuronales competitivas en tareas de cuantización vectorial, pero en las que se incluye una función de magnitud objetivo. El efecto de ésta función es forzar a las unidades a concentrarse en zonas de alto valor de la magnitud, calculada localmente a partir de los datos o de los parámetros de cada unidad. En éste sentido MSCNNs difiere de otros algoritmos de aprendizaje competitivo que usualmente generan una aproximación a la densidad de probabilidad de los datos de entrada. Por tanto MSCNNs son mucho más versátiles al ser capaces de distribuir los prototipos siguiendo otras propiedades o características de los datos.

Los ejemplos de aplicación mostraron la capacidad de MSCNN en distintas aplicaciones: Cuantización de distribuciones Gausianas, interpolación de series de datos, modelado de superficies, cuantización de color o compresión selectiva de imágenes. Los resultados comparando los nuevos algoritmos frente a otros algoritmos competitivos validaron las ventajas de las redes MSCNNs en aquellas tareas en los que el la distribución deseada de las unidades no se corresponde con la densidad de distribución de los datos.

Las mayores contribuciones de la tesis son:

1. Se ha propuesto un algoritmo competitivo de tipo 'hard competitive', MSCL (descrito en [63] y explicado exhaustivamente en [61]) con la propiedad de distribuir los centroides en zonas del espacio siguiendo una magnitud arbitraria calculada localmente para cada unidad. Dicho algoritmo se basa en dos métodos para conseguir su objetivo: Uso de dos fases de competición y hecho del empleo de un factor de aprendizaje distinto para cada neurona. El algoritmo se ha propuesto en dos implementaciones,

secuencia y por lotes.

2. Se ha propuesto un algoritmo competitivo de tipo 'soft competitive' inspirado en los Mapas autoorganizados de Kohonen, MS-SOM (descrito en [65]). Este algoritmo también tiene la propiedad de distribuir los centroides en zonas del espacio siguiendo una magnitud, pero lo hace conservando las relaciones de vecindad entre las unidades, con lo que se consigue una representación discreta del espacio de entrada (normalmente bidimensional).

3. Se ha presentado una versión enmascarada da MSCL y MSSOM que permite el aprendizaje a partir de datos de distinta longitud (la idea se esbozó en (64)). Este algoritmo es útil en situaciones en las que alguna de las muestras tiene componentes inválidas, por ejemplo en el modelado de procesos industriales, donde falla algún sensor, o en el procesamiento de datos estadísticos procedentes de distintas fuentes. Un algoritmo convencional falla en éstas situaciones ya que requiere algún tipo de preprocesamiento artificial de los datos para igualarlos y poder tratarlos durante el entrenamiento.

4. Se ha presentado una generalización de dos algoritmos usados habitualmente para la inicialización de K-means: KKZ y K-Means++. El nuevo algoritmo, MS-INIT emplea un factor de magnitud adicional (asociado a cada muestra) durante la selección de las unidades. Se ha demostrado que el nuevo algoritmo produce una mejor inicialización de un codebook para luego ser entrenado mediante MSCL.

5. También se ha desarrollado un nuevo algoritmo de compresión selectiva de imágenes, MSIC ([64]), que emplea diversas redes neuronales de tipo MSCL y que consigue que distintas áreas de una imagen reciban distinto ratio de compresión dependiendo de la saliencia de la imagen. El algoritmo se basa en dividir la imagen en bloques irregulares de distinto tamaño de acuerdo a dicha saliencia, que luego son comprimidos con MSCL enmascarado. Los experimentos han demostrado las ventajas de MSIC frente a JPEG y SOM en compresión selectiva de imágenes.

Otras contribuciones menores de la tesis son las siguientes:

1. Se han definido dos medidas de calidad de cuantización aplicables a redes de tipo MSCNN: WMSE y Weighted Entropy, ambas versiones ponderadas por la magnitud del 'Mean Squared Error' y de la entropía.

2. También se ha demostrado la ventaja de MSCL y MSSOM en modelado de superficies (tanto de imágenes de profundidad 3D como de nubes de puntos de scanner láser) ya que los nuevos algoritmos son capaces de centrar las neuronas en zonas de alta curvatura.

3. Se han mostrado las ventajas de los nuevos algoritmos en una tarea de reducción de colores. En concreto se ha buscado que la paleta reducida de colores enfatizase aquellos que se encontrasen en zonas de mayor saliencia (para lo cual se han definido distintas medidas de saliencia).

   Finalmente se ha comparado SOM y MS-SOM en tareas de clasificación de datos. Se ha seleccionado una magnitud que fuerza las neuronas a centrarse en las zonas de mayor error de clasificación. De éste modo el error de clasificación final de la red entrenada ha sido menor empleando MS-SOM que el obtenido cuando se ha empleado SOM.

Part V

APPENDICES

Appendix A

Abbreviations

Name	Meaning
ANN	Artificial Neural Network
BMU	Best-Matching Unit
CL	Competitive Learning
CR	Color reduction
CQ	Color Quantization
FSCL	Frequency Sensitive Competitive Learning
MSCL	Magnitude Sensitive Competitive Learning
MSCNN	Magnitude Sensitive Competitive
MSIC	Magnitude Sensitive Image Compression
MS-INIT	Magnitude Sensitive initialization
MS-SOM	Magnitude Sensitive Self-Organizing Map
NG	Neural Gas
k-NN	k-Nearest Neighbors
SGONG	Self-Growing and Self-Organized Neural Gas
SOM	Self-Organizing Map
VQ	Vector Quantization

Table A.1: List of abreviations

Appendix B

MS Toolbox

B.1 Introduction

This technical report presents the Magnitude Sensitive Toolbox (MS Toolbox) hereafter simply called the Toolbox, for Matlab computing environment by MathWorks, Inc.

The Toolbox comprises a number of useful functions to use the MSCL and the MS-SOM algorithms. It uses some functions from the SOM Toolbox, developed by the Laboratory of Computer and Information Science (CIS) at the Helsinki University of Technology. This Toolbox is downloadable from `http://www.cis.hut.fi/somtoolbox/`

MSCL, as other unsupervised learning algorithms may be trained online, which have the advantage of avoiding an extensive preprocessing of the dataset or the storage of this information in long temporal memories. However, due of the fact that it is necessary to update the value of the magnitude, it might require excesive computation time if made for each new sample.

That is why we decided to work in epochs with both MSCL and MS-SOM algorithms. Input data is divided in small blocks (or if received online, saved in small memories). Then, at every epoch, a data block that we call `dataep` is presented to the MSCL neural network.

The MSCL is then trained with `dataep`: Unit weights compete first in a global competition considering only the distance of each unit to the samples in `dataep` and in a second step, competition is local, but it haves into account the value of the magnitude, $mg_i(t)$. Then, weights of winner units are updated. These steps, unit competition and update, might be done in two ways:

- Sequentially, where unit competition and update is done just after the presentation of each individual sample in `dataep`

- In batch mode, where unit competition is done with all samples in `dataep` and after that all winner's weights are updated.
- 'Mask' mode, where unit competition and update is done just after the presentation of each individual sample in `dataep`, but each sample have associated a mask (different for each sample) used to determine the components used in that sample during the competition.

The magnitude associated to each unit is calculated by calling a user magnitude function `magfunct()`, as it is explained in section B.3, and then updated.

Section B.2 explain the structures and the different functions used in the algorithm. Section B.3 explain how to define and use a magnitude function, and gives several examples of useful magnitude functions. Finally last section explain some demos.

The MS-Toolbox is available free of charge under the GNU General Public License from: http://www.researchgate.net/profile/Enrique_Pelayo

B.2 MS Toolbox

This section gives an explanation of the Toolbox. Additional information is available from the help section of every function.

The kind of data that can be processed with the Toolbox is so-called spreadsheet or table data. Each row of the table is one data sample. The columns of the table are the variables or components of the data set. The variables might be the properties of an object, or a set of measurements at an specific time. The Toolbox can handle only numeric data. Every sample may have the same number of components (that we call `DIM`), or have different number. In this case it is used the 'Mask' training mode . If the available data do not agree with these specifications, they can usually be transformed so that they do.

The total number of samples in the dataset is called `LENGTH`. These data comprises one cycle. During training it is possible to repeat the presentation of the whole dataset if a number of cycles greater than one is selected. As it has been previously mentioned, dataset is divided in training epochs (Each cycle is divided in several epochs). The subset of the data that correspond to an epoch is `dataep` and we call its length: `epochlength`.

B.2.1 Structures

The Toolbox includes two structures to group related information. The first one (the MS struct) is the neural network. The second one is a structure created to pass the parameters

to the magnitude function: the Magnitude Function struct. The MS-SOM also uses an additional MS-struct to train the SOM map (it is used to define several MS-SOM training parameters).

The MSCL struct includes all the information about the neural network. It is initialized by the *mscl_init*() function. Its fields are:

Field	*Type*	*Description*
.name	(string)	Name of the neural network
.type	(string)	Fixed value: 'ms_struct'
.cb	(num_units x DIM)	Codebook matrix. Units expressed in rows
.mu	(vec., num_units x 1)	Magnitudes associated to each unit
.macc	(vec., num_units x 1)	Accumulated magnitude in each unit
.conn	(num_units x num_units)	Sparse connexion matrix
.*train*	(struct)	Training parameters:
.msk	(vec., DIM)	Component mask vector
.wins	(integer)	Number of competing units in local mode
.gamma	(integer)	Gamma
.beta	(integer)	Beta
.kconns	(integer)	Forgetting parameter
.function	(@pointer)	Pointer to the magnitude function
.args	(cell)	Arguments
.*flags*	(struct)	-
.acc	(boolean)	Use accumulated magnitude
.conn	(boolean)	Calculate connexions
.datafcn	(boolean)	Magnitude defined by sample input
.unitfcn	(boolean)	Magnitude defined by unit weights
.cmdfcn	(boolean)	Magnitude defined by unit commands
.update	(boolean)	Update weights and magnitudes
.*cmds*	(struct)	Command flags corresponding to the optional MF struct fields. If one flag, the corresponding field of a MF struct is filled.

Table 1. MS struct.

The Magnitude Function struct (MF struct) is created and filled within the different training functions, and it is used to pass the parameters to the Magnitude Function. This function only may receive an argument, the MF struct. Its fields are:

Field	*Type*	*Description*
.maskbmu	(vec., DIM)	Global component mask.
.args	(cell)	User arguments.
OPTIONAL:		*Optional fields*
.mu	(vec., num_units x 1)	Magnitudes associated to each unit.
.mg	(vec., num_units x 1)	Magnitudes used in BMU local competition.
.macc	(vec., num_units x 1)	Accumulated magnitudes in each unit.
.qerr	(epochlength x 1)	Quantization error (samples at the epoch).
.bmus	(epochlength x 1)	Best Matching Units (samples at the epoch).
.dataep	(epochlength x DIM)	Data matrix (samples at the epoch).
.conns	(num_units x num_units)	Connection matrix.

Table 2. MF struct.

It is important to remark that the optional fields are only created depending on the parameters provided when passing the magnitude function pointer to the training functions (see section B.3).

B.2.2 Initialization and training functions

There are one initialization and three training implementations of the MSCL algorithm (sequential, sequential with masked components and batch [*default*]) in the Toolbox, and two for the MS-SOM algorithm (sequential [*default*] and sequential with masked components).

The simplest way to initialize and train a MSCL is to use function `ms_make()` which does both by default using automatically selected parameters. Depending on these parameters, it is possible also to define that the MS struct is only created and defined with initial values, or on the contrary, that it is trained a previously existing MS neural network.

The function divides the dataset in epochs what are passed to one of these low level training functions:

- `ms_seqtrain(sM, data)` : Trains sequentially a MSCL network.
- `ms_batchtrain(sM, data)` : Training is done in batch mode.
- `ms_masktrain (sM, data, datamask)` : Training is done in sequential mode, but using an associated a mask for each sample.

Additionally there are two functions for initialization and training. Both functions directly call `ms_make()` and are equivalent to use it with the same parameters:

- `ms_init(num_units, data, [[argID, value],...])` : Creates and initializes a MSCL network. It is equivalent to the use of:

  `ms_make(num_units, data, 'init', [[argID, value],...])` .

- `ms_train(MS, data,[[argID, value],...])` : Trains a MSCL network. It is equivalent to the use of `ms_make(MS, data,[[argID, value],...])` .

On the other hand, MS-SOM training is achieved by `ms_somtrain(sMap, sM, D, [[argID, value],...])` . This function use sequential algorithm to train a Magnitude Sensitive Self-Organizing Map (MS-SOM) depending on unit magnitude. Unit magnitude and some training parameters related to the use of magnitude are passed through a MS-Struct. The function returns both trained structures (codebooks for both are the same).

B.2.3 Visualization functions

There are two functions to visualize trained MS networks (MSCL or MS-SOM): `ms_fig()` and `ms_connplot()`:

- `ms_fig(Dw, nns, [[argID, value],...])`: The first one draws dataset (up to 3 selected dimensions), and unit centers of a MS struct (with same number of components). It is possible to select which internal variable of each unit is represented with a colour palette (.mu, .macc, ...). This function is also capable of drawing the SOM grid if the network corresponds to a MS-SOM (in this case it is necessary to pass both, the MS and the SOM structs).

- `ms_connplot(sM, [[argID, value],...])`: The second function is used to draw the connection matrix obtained during or after training. It is only valid for MS-structs (and it is equivalent to the SOM grid).

B.2.4 Auxiliary functions

This is the list of other auxiliary functions in the MS-Toolbox:

- `ms_connmake(sM,data)`: Generates the connection matrix of a MSCL.

- `ms_cbinit(numunits, datainput, [[argID, value], ...])` : Competitive algorithm for generating an initial codebook for a MSCL neural network. Centers are selected to minimize the sum of weighted squared distance from each center to the total of data samples within its Voronoi region.
- `ms_bmus(sM, data)` : Find the best-matching units from the MSCL for the given vectors, having into account the magnitude at each unit.
- `ms_bmusmask(sM, data, mask)` : Find the best-matching units from the MSCL for the given vectors, having into account the magnitude at each unit. Use individual masks for each data sample.
- `ms_getcmds(arguments)` : Extracts the commands from the 'arguments' cell array (containing command strings and arguments for the magnitude function), sets command flags corresponding to the optional MF struct fields, and extracts arguments in an isolated cell array. The output results are used when calling a magnitude function that uses a MF struct. It is possible that 'arguments' contains no string commands. In that case, *iscmf* flag is set to FALSE and input is passed directly to 'args'.
- `ms_beta(cycles, numdata, numunits, finalalpha)` : Calculates the value of beta, optimized so the final alpha value tends to a user defined value. If this value is null, it is optimized to the value that it would get in one cycle.
- `ms_quality(sM, data)` : Calculates the Weighted Mean Squared Error (WMSE) and Weighted Entropy of a trained MSCL.

B.3 The magnitude function

B.3.1 Use of the magnitude function

MSCL and MS-SOM algorithms need to define the value of the magnitude at each unit for the competition step. The magnitude is a scalar, that can be calculated through the magnitude function mf. This function takes the general form:

$$mf : Y \to \mathbb{R} \tag{B.1}$$

There are three ways to define it, depending on which arguments (Y) are used:

1. As a function of data sample $\mathbf{x}$. A particular case is when this function can not be explicitly defined, but there is still a scalar value that can be assigned to that sample,

$mx(t)$. In this case it may be created an extended data vector from the concatenation of $\mathbf{x}$ and $mx(t)$:

$$\mathbf{x}_{extended}(t) = [\mathbf{x}(t)\, mx(t)] \tag{B.2}$$

Then, this vector is used instead of $\mathbf{x}$.

2. As a function of the values of each unit weights $\mathbf{w}_k(t)$.

3. As a function of the values of each unit weights $\mathbf{w}_k(t)$ and other parameters of the Voronoi region corresponding to unit k.

To use the *MS Toolbox* the user may develop its own function to define the magnitude function. This function always outputs two vectors:

- *mags*: new values of the magnitudes for each unit may be calculated from the data samples (matrix) at the current epoch, or the codebook matrix (depending on the input arguments). Its length corresponds to the matrix number of rows, and returns a scalar for each row.

- *valid*: vector of flags that indicates if the corresponding magnitude is a valid value.

However, the input arguments are different depending on the way to define the function:

1. In the first case, Y is formed by a data matrix (with data in rows), and additional function arguments, $args$ (cell if more than one, or a numeric scalar, vector or an array if it is a single argument):

$$[mags\, valid] = user\_mf(X, args) \tag{B.3}$$

in this case outputs are two column vectors with size equal to the dataset length.

2. In the second case, Y is formed by the codebook matrix (with units in rows), and the additional function arguments (cell if more than one, or a numeric scalar, vector or an array if it is a single argument):

$$[mags\, valid] = user\_mf(codebook, args) \tag{B.4}$$

in this case outputs are two column vectors with size equal to the number of units.

3. In the last case, Y is formed by the codebook matrix (with units in rows), and the structure function `MF`. It is a structure described before whose fields are created and modified by training functions. These fields may be the data in the epoch (`dataep`), several training variables (bmus, hits, quantization error,...) or other user defined arguments:

$$[mags\, valid] = user\_mf(codebook, MF) \tag{B.5}$$

   In this case, outputs are also two vectors with size equal to the number of units.

It is important to remark that it is possible the situation where one o more units have not valid value for *mags*. One example of this situation is when one unit k has no hits in one epoch, but the magnitude function uses the data associated to the unit. In that case valid(k) indicates that mags(k) can not be calculated.

The way to use it is through the `'magfunction'` or `'unitfunction'` parameter used when calling any of the initialization or training functions, for both MSCL and MS-SOM. It is necessary to indicate which of the fields will be necessary for the function, so the training function can fill the MF struct fields.

This parameter is always followed by a pointer to the user function and a cell with a list of fields names and the user arguments:

$$...,'\texttt{function}', @\texttt{user\_mf}, \{\texttt{commands}, \texttt{userargs}\}), ...$$

where `commands` is the list of field names and `userargs` are the user arguments.

For example, the function `user_mf1` is called by `ms_make()` in the following way:

$$\texttt{MS} = \texttt{ms\_make}(\texttt{data}, \texttt{MSini}, '\texttt{function}', @\texttt{user\_mf1}, \{'\texttt{qerr}', '\texttt{bmus}', \texttt{arg1}, \texttt{arg2}\});$$

Then it initializes a MS struct defining the `.train.fcn` field to be equal to the pointer to `user_mf1`, and additionally it forces to create a MF struct with the following fields:

`.maskbmu:` mask vector.

`.args:` { argument1 argument2}.

`.qerr:` the quantization error of each sample of the epoch data.

`.bmus:` the best matching unit of each sample of the epoch data.

This makes it possible to use any of these variables internally into the user function :

```
function [ mags, valid ] = user_mf1(codebook, MF)
```

```
   mask = MF. maskbmu; % mask
   numberunits = size(codebook,1);
   argument_one = MF.args{1}; % arguments
   argument_two = MF.args{2};
   quantizationerrors = MF.qerr;
   ...
end
```

B.3.2 List of magnitude function examples

Here you can find a list of useful pre-programmed magnitude functions. They are expressed in the way that they must be called in the initialization or training functions. Parameters between single quotes are required and must be expressed as they are. User parameters may also be necessary. In that case, they are written in italic.

- `..., @msf_ones, [],...`: Returns the constant magnitude value '1' for each unit.
- `..., @msf_dist, {points, distmax}, ...`: Returns the minimum distance from each sample (in rows) to a set of *points*.
- `..., @msf_xcomp, comp, ...`: Returns a column vector of the absolute value of the selected input *comp*.

B.4 Demos

- `demo_mscl.m`: Test of several initialization an training modes of a MSCL neural network. Uses simple magnitude functions.
- `demo_quality.m`: Test Entropy and WMSE calculation during an epoch and at the end of the algorithm.
- `demo_figs.m`: Test of visualization functions.
- `demo_mask.m`: Test of initialization an training of MSCL and MS-SOM neural networks in 'mask' mode. Uses simple magnitude functions.
- `demo_mssom.m`: Test of several initialization an training modes of a MS-SOM neural network. Uses simple magnitude functions.

Bibliography

[1] S. C. Ahalt, A. K. Krisnamurthy, P. Chen, and D. E. Melton, Competitive learning algorithms for vector quantization, Neural Networks 3 (1990) 277-290.

[2] N. Ahmed, T. Natarajan, K.R. Rao: Discrete cosine transform. Computers, IEEE Transactions on **100**(1) (1974) 90–93

[3] D. Alahakoon, S. K. Halgamuge, B. Srinivasan, Dynamic self-organizing maps with controlled growth for knowledge discovery. Neural Networks, IEEE Transactions on, 11:3, (2000) 601-614.

[4] N. Amenta, M. Bern, M. Kamvysselis, A new voronoi based surface reconstruction algorithm, in: Proc. of Siggraph 98 (1998) 415-422.

[5] C.Amerijckx, J.D. Legat, M. Verleysen: Image compression using self-organizing maps. Systems Analysis Modelling Simulation **43**(11) (2003) 1529–1543

[6] H. Annuth and C.A. Bohn, Smart Growing Cell: Supervising Unsupervised Learning, Studies in Computational Intelligence 399 (2012) 405-420.

[7] S. Arias, H. Gómez, F. Prieto, M. Botón, R. Ramos: Satellite image classification by self organized maps on GRID computing infrastructures. In Proceedings of the second EELA-2 Conference, pp. 1-11. (2009)

[8] D. Arthur, S. Vassilvitskii, K-means++: the advantages of careful seeding. In: ACM-SIAM Symposium (2007)

[9] A. Atsalakis, and N. Papamarkos, Color reduction and estimation of the number of dominant colors by using a self-growing and self-organized neural gas. *Engineering Applications of Artificial Intelligence*, (2006) 19(7):769–786.

[10] J. Bezdek, *Pattern recognition with fuzzy objective function algorithms*. Plenum Press, (1981).

[11] C. M. Bishop, M.Svensen and C. K. I. Williams, GTM: The generative topographic mapping, Neural Computation 10 (1998) 215-234.

[12] M. Celebi, An effective color quantization method based on the competitive learning paradigm, in: Proceedings of the International Conference on Image Processing, Computer Vision, and Pattern Recognition (2009), 876-880.

[13] M. Celebi and G. Schaefer, G. Neural Gas Clustering for Color Reduction, in Proceedings of the International Conference on Image Processing, Computer Vision, and Pattern Recognition (2010) 429-432.

[14] M. Celebi, Improving the Performance of K-Means for Color Quantization. *Image (Rochester, N.Y.)*, (2011) 29(4):260–271.

[15] G. J. Chappell, J. G. Taylor, The temporal Kohønen map. Neural networks, 6:3, (1993) 441-445.

[16] C. Chang, P. Xu and R. Xiao, New adaptive color quantization method based on self-organizing maps. *Neural Networks, IEEE*, (2005) 16(1):237–249.

[17] G. Cheng, Image Color Reduction Based on Self-Organizing Maps and Growing Self-Organizing Neural Networks, in: proceedings of the Sixth International Conference on Hybrid Intelligent Systems,(2006), 24.

[18] Y. Cheung, On rival penalization controlled competitive learning for clustering with automatic cluster number selection, IEEE Transactions on Knowledge Data Engineering 17 (2005) 1583-1588.

[19] D. Cutting, D.Karger, J. Pedersen, J.Scatter/Gather, A Cluster-based Approach to Browsing Large Document Collections. In: ACM SIGIR Conference (1992)

[20] D.L. Davies, D.W. Bouldin, A cluster separation measure, IEEE Transactions On Pattern Analysis and Machine Intelligence (1979) 224-227.

[21] A. Dekker ,Kohonen Neural Networks for Optimal Colour Quantization, Network: Computation in Neural Systems 3:5 (1994) 351-367.

[22] A. P. Dempster, N.M. Laird, D.B. Rubin, Miximum Likelyhood from Incomplete Data via the EM Algorithm, Journal of the Royal Statistical Society, Series B, 39(1) (1977) 1-38.

[23] N. Dhavale, L. Itti, Saliency-Based Multi-Foveated MPEG Compression, in: Proceedings of the IEEE Seventh International Symposium on Signal Processing and its Applications, (2003) 229-232.

[24] R. Durbin, and D. Willshaw, An analogue approach to the travelling salesman problem using an elastic net method, Nature 326 (1987) 689-691.

[25] http://visionair.ge.imati.cnr.it/ontologies/shapes/.

[26] M. Harandi, M. Gharavi-Alkhansari: Low bitrate image compression using self-organized kohonen maps. In: Proceedings 2003 International Conference on Image Processing, ICIP'03. Volume 3. (2003) 267-270

[27] J. Harel, C. Koch, and P. Perona, Graph-Based Visual Saliency, in: Proceedings of the NIPS 2006, (2006) 545-552.

[28] http://www.klab.caltech.edu/~harel/share/gbvs.php.

[29] H. Hoppe, T. Derose, T. Duchamp, J. Mcdonald, W. Stuetzle, Surface reconstruction from unorganized points, in : Proc. of Siggraph 92 (1992) 71-78.

[30] Computer Vision Group, U.o.G.: Dataset of standard 512x512 grayscale test images. http://decsai.ugr.es/cvg/CG/base.htm (2002)

[31] Iris Dataset, http://archive.ics.uci.edu/ml/datasets/Iris

[32] L. Itti, C. Koch, and E. Niebur, A model of saliency based visual attention for rapid scene analysis, IEEE Transactions on Pattern Analysis and Machine Learning 20 (11) (1998) 1254-1259.

[33] L. Itti, and C. Koch Computational modeling of visual attention. *Nature reviews neuroscience*, (2001) 2(3), 194-203.

[34] C. Koch and S. Ullman, Shifts in selective visual attention: towards the underlying neural circuitry. *Human Neurobiology*, (1985) 4(4), 219-227.

[35] I. Ivrissimtzis, W.K. Jeong, S. Lee, Yunjin Lee, H.P. Seidel, Surface Reconstruction Based on Neural Meshes, in proceedings Mathematical Methods for Curves and Surfaces, (2005) 223-242.

[36] I. Katsavounidis, C. Kuo, Z. Zhang, A new initialization technique for generalised lloyd iteration. IEEE Signal Processing Letters 1 (10), (1994) 144-146.

[37] K. Kerdprasop, N. Kerdprasop, P. Sattayatham, Weighted K-Means for Density-Biased Clustering, in: Proceedings of the Data Warehousing and Knowledge Discovery, 7th International Conference (DaWaK 2005) 488-497.

[38] T. Kohonen. Self-Organizing Maps, Springer-Verlag, New York, 1997.

[39] P. K. Kuhl, Human adults and human infants show a 'perceptual magnet' effect for the prototypes of speech categories, monkeys do not, Perception and Psychophysics 50 (1991) 93-107.

[40] P. K. Kuhl, K. A. Williams et al. , Linguistic experience alters phonetic perception in infants by 6 months of age, Science 255 (1992) 606-608.

[41] A. Laha, N. Pal, B. Chanda, Design of vector quantizer for image compression using self-organizing feature map and surface fitting. Image Processing, IEEE Transactions on (2004) **13**(10) 1291-1303

[42] J. Lampinen, E. Oja, Clustering properties of hierarchical self-organizing maps. Journal of Mathematical Imaging and Vision, 2:2-3, (1992) 261-272.

[43] J. Lazaro, J. Arias, J. Martin, A. Zuloaga, and C. Cuadrado, SOM Segmentation of gray scale images for optical recognition. *Pattern Recognition Letters*, (2006) 27(16) 1991-1997.

[44] A. Likas, N. Vlassis, J.V. Jakob, The global k-means algorithm. Pattern Recognition 36(2), 451:461 (2003)

[45] R.J. Liou, Image compression using sub-band dct features for self-organizing map system. Journal of Computer Science and Application (2007) **3**(2)

[46] Y. Liu, R. H. Weisberg, Patterns of ocean current variability on the West Florida Shelf using the self-organizing map. Journal of Geophysical Research, (2005) 110(C6), C06003.

[47] Y. Liu, R. H. Weisberg, C. N. Mooers, Performance evaluation of the self-organizing map for feature extraction. Journal of Geophysical Research: Oceans, (2006), 111(C5), 1978-2012

[48] S. Lloyd, Least Squares Quantization in PCM, IEEE Transactions on Information Theory 28(2) 129-136 (1982).

[49] J. MacQueen, Some Methods for Classification and Analysis of Multivariate Observations, Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability, (1967) 281-297.

[50] Th. M. Martinetz, S. G. Berkovich, and K. J. Schulten, 'Neural-gas' network for vector quantization and its application to time-series prediction, IEEE Trans. on Neural Networks 4 (1993) 558-569.

[51] P. Martinez,et al., Hyperspectral image classification using a self-organizing map. In Summaries of the X JPL Airborne Earth Science Workshop (2001).

[52] Matlab, from MathWorks http://www.mathworks.es

[53] M. Melato, B. Hammer , K. Hormann, Neural Gas for Surface Reconstruction, Technical reports, Clausthal-Zellerfeld: Clausthal University of Technology.(2007).

[54] E. Merenyi, A. Jain, T. Villmann, Explicit magnification control of self-organizing maps for 'forbidden' data. IEEE Transactions on Neural Networks (2007) 18(3): 786-797.

[55] A. Nachter, K. Lingemann, J. Hertzberg, H. Surmann, 6D SLAM 3D Mapping Outdoor Environments, Journal of Field Robotics 24(8/9) (2007) 699-722.

[56] N. Nikolaou and N. Papamarkos, Color reduction for complex document images, International Journal of Imaging Systems and Technology 19(1) (2009) 14-26.

[57] T. Onoda, M. Sakai, S.Yamada, Independent Component Analysis based Seeding method for k-means Clustering. In: IEEE/WIC/ACM Conference (2011).

[58] N. Otsu, A threshold selection method from gray-level histograms, IEEE Transactions on Systems, Man, and Cybernetics, 9(1) (1979) 62-66.

[59] N.Papamarkos, A neuro-fuzzy technique for document binarisation. *Neural Computing and Applications*, (2003), 12(3-4):190–199.

[60] E. Pelayo, J. D. Buldain, C. Orrite, SO-VAT: Self-Organizing Visual Assessment of cluster Tendency for large data sets, in proceeding of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN (2011) 141-146.

[61] E. Pelayo, D. Buldain, C. Orrite, Magnitude Sensitive Competitive Learning. Neurocomputing 112, (2013) 4-18.

[62] E. Pelayo, D. Buldain, C. Orrite, Focused Image Color Quantization using Magnitude Sensitive Competitive Learning Algorithm, in Proceedings of the 4 th International Joint Conference on Computational Intelligence, IJCCI, (2012) 516-521.

[63] E. Pelayo, J. D. Buldain, C. Orrite, Magnitude Sensitive Competitive Learning, in proceeding of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN (2012) 305-310.

[64] E. Pelayo, J. D. Buldain, C. Orrite, Magnitude Sensitive Image Compression, in Proceedings of the 5 th International Joint Conference on Computational Intelligence, IJCCI, (2013) 370-380. (*This paper received the NCTA 2013 - BEST STUDENT PAPER AWARD*).

[65] E. Pelayo, J. D. Buldain, MS-SOM: Magnitude Sensitive Self-Organizing Maps, in Proceedings of the 10th Workshop On Self-organizing Maps, WSOM (2014) (*Review pending*).

[66] N. Ponomarenko , S. Krivenko , K. Egiazarian J. Astola, V. Lukin, Weighted MSE Based Metrics for Characterization of Visual Quality of Image Denosing Methods, Fifth International Workshop on Video Processing and Quality Metrics for Consumer Electronics, Scottsdale, Arizona USA, (2010).

[67] L. Prechelt, Proben1 - a set of neural network benchmark problems and benchmarking rules. Technical Report 21/94, Fac. of Informatics, Univ. Karlsruhe (1994).

[68] R. do Rego, F.de Lima Neto, Growing Self-Organizing Maps for Surface Reconstruction from Unstructured Point Clouds, in: Proc. of theInternational Joint Conference on Neural Networks (2007) 1900-1905.

[69] U. Rutishauser, D. Walther, C. Koch, P. Perona, Is bottom-up attention useful for object recognition, in: Proceedings of the CVPR 2004, Vol. 2 (2004) 37-44.

[70] K. Tasdemir, E.Merenyi, Considering topology in clustering of the Self-Organizing Maps, in: Proc. of 5^{th} Workshop on Self-Organizing Maps (2005) 439-446.

[71] A. Treisman and G.Gelade, A feature integration theory of attention. *Cognitive Psychology*,(1980), 12:97–136.

[72] H. Shah-Hosseini, R. Safabakhsh, TASOM: a new time adaptive self-organizing map. Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, 33:2, (2003) 271-282.

[73] http://www-graphics.stanford.edu/data/3Dscanrep/.

[74] H. Song and HY. Feng, A Point Cloud Simplification Algorithm for Mechanical Part Inspection, Int J. Adv Manuf Technol 45 (2009) 583-592.

[75] T. Uchiyama, and M. Arbib, Color image segmentation using competitive learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (1994), 16(12), 1197-1206.

[76] DGait Database, http://www.cvc.uab.es/DGaitDB/Summary.html.

[77] E. Vazquez, T. Gevers, M. Lucassen, J. van de Weijer, and R. Baldrich, Saliency of color image derivatives: a comparison between computational models and human perception. *Journal of the Optical Society of America. A, Optics, image science, and vision*,(2010), 27(3) 613-21.

[78] M. Varstal, J. D. R. Millán, J. Heikkonen. A recurrent self organizing map for temporal sequence processing. In Artificial Neural Networks, ICANN'97 . Springer Berlin Heidelberg, (1997) 421-426.

[79] J. Vesanto, J. Himberg, E. Alhoniemi and J. Parhankangas, Self-organizing map in Matlab: the SOM Toolbox, in: Proceedings of the Matlab DSP Conference 1999, (1999) 35-40.

[80] T. Villmann, E. Merényi, B. Hammer, Neural Maps in Remote Sensing Image Analysis. Neural Networks, Special Issue on Self-Organizing Maps for Analysis of Complex Scientific Data, 16:(3-4) (2003) 389-403.

[81] J. C. Claussen, Magnification Control in Self-Organizing Maps and Neural Gas, Neural Computation, 18 (2006) 446-469.

[82] CD. Wang, JH. Lai, Energy based competitive learning, Neurocomputing 74 (2011) 2265-2275.

[83] L. Xu, A. Krzyzak, and E. Oja, Rival Penalized Competitive Learning for Clustering Analysis, RBF net and Curve Detection. *IEEE Tr. on Neural Networks*, (1993), 4 636-649.

[84] H. Yin, Learning nonlinear principal manifolds by self-organising maps. In Principal Manifolds for Data Visualization and Dimension Reduction. Springer Berlin Heidelberg (2008) 68-95.

[85] Y. Yu, Surface reconstruction from unorganized points using self-organizing neural networks, in: Proc. of IEEE Visualization Conference (1999) 61-64.

[86] B. Zhang, Generalized k-harmonic means - boosting in unsupervised learning. Technical Report HPL-2000-137. Hewlett-Packard Labs (2000)