Source Coding - Codifica di Sorgente
Project 6

Tommy Azzino - 1151517

Abstract

Nowadays, most of the time we face limited resources such as bandwidth in communications and storage memory in our devices. Therefore, when dealing with image storage or transmission over the most common communication media, one extremely important factor is image compression. Thanks to compression it's possible to obtain images of manageable sizes that suit channel conditions in communication or the amount of available memory in storage application. Among different techniques belonging to *lossless* and *lossy* categories, the Linde, Buzo and Gray (LBG) algorithm, introduced in 1980, is a vector quantization algorithm with the aim of deriving a good codebook over a set of training images. With this intent, one key aspect is codebook initialization. In this project, a review of vector quantization and of the LBG algorithm is carried, then a performance comparison between different initialization techniques and vector sizes is conducted. Moreover, the goodness of an initialization method based on the extraction of initial codebook vectors using an image's pyramid is investigated. Results are evaluated over the training and test sets in terms of Peak Signal-to-Noise Ratio (PSNR).

(i) Introduction

A fundamental factor in image storage or transmission is image compression. All image compression techniques fall in two main categories: *lossless* and *lossy*. In the former set of methods, an image can be reconstructed after compression without any loss of data during the process. Whereas in the latter, the compression is irreversible, since quantization is involved. On the other hand *lossy* encoding can achieve higher compression rate at the expense of some distortion introduction. Typically, in *lossy* procedures quantization is exploited. It can be *scalar* quantization: where each input data is quantized independently from the others, or *vector*: where a block of input samples is jointly quantized. One optimal algorithm for Vector Quantization (VQ) is the Linde, Buzo and Gray (LBG) algorithm. Objective of the LBG algorithm is the construction of an optimal codebook constituted by the most representative feature vectors over a set of training images. Then, the set of representative codevectors can be used in order to perform VQ. Specifically, the entire VQ procedure can be divided in three main steps:

- **1.** Optimal codebook computation.
- **2.** Image encoding through the computed codebook.
- **3.** Image decoding at receiver side.

The first step is the most important since our objective is to build a codebook that minimizes the overall distortion introduced by quantization. Another important consideration is related to the choice of training set. It has to be representative about the type of images that need to be compressed. In the second step, the image is transformed into vectors. Subsequently, each vector is replaced with the index of its closest codevector in the codebook and only the indexes for each vector are sent over the channel, thus reducing the required bit rate (or the required memory in storage applications). Therefore, at receiver side we need to hold a copy of the codebook. Indeed in the final phase, each vector is approximated with the codevector that corresponds to the received index, and the entire *lossy* image is decoded.
The reminder of this report is organized as follows: in section (ii) a review of VQ and a detailed explanation of LBG algorithm are given, while in section (iii) the simulation environment and details about the code structure are presented. In section (iv) a performance analysis of LBG is conducted. Moreover, a non-standard initialization approach is discussed and it's performance evaluated against other classic methods. Finally in section (v), a resume of the work done and final considerations are carried out.

(ii) VQ and LBG Algorithm

Vector Quantization

Considering blocks of symbols rather than individual symbols has shown to improve efficiency in the field of entropy encoding. As a consequence, this strategy can be also applied in quantization. Vector quantization differs form scalar quantization since in the former we quantize a set of samples instead of a single value each time. Formally, this procedure can be described as follows: given a dimension N, we define an input vector (to be quantized) as:

$$\mathbf{x} = [x_1, x_2, ..., x_N], \quad \mathbf{x} \in \mathbb{R}^N \tag{1}$$

where each x_i with $i = 1, 2, ..., N$ represents a single sample. The set of quantization (or reconstruction) vectors, called codebook (C), of size K is defined as:

$$C = \{\mathbf{c}_1, \mathbf{c}_2, ..., \mathbf{c}_K\} \tag{2}$$

where each $\mathbf{c}_i$ with $i = 1, 2, ..., K$ is a codevector. The quantization procedure assigns an input vector to its closet codevector in the codebook and therefore a vector quantizer can be described by a set of *decision regions* or *cells* that constitute a partition of the N-dimensional real space. Mathematically, these regions are defined as follows:

$$I_i, \quad i = 1, 2, ..., K : \quad I_i \cap I_j = \emptyset \quad \forall i \neq j \quad and \quad \cup_{i=1}^{K} I_i = \mathbb{R}^N \tag{3}$$

Straightforward is the definition of quantization *rule*:

$$\mathbf{Q}(\mathbf{x}) = \mathbf{c}_i \quad if \quad \mathbf{x} \in I_i \tag{4}$$

The quality of a vector quantizer is assessed by average distortion measure between the input $\mathbf{x}$ and the quantizer's output $\mathbf{c} = \mathbf{Q}(\mathbf{x})$. The most used distortion measure is the following squared error:

$$d(\mathbf{x}, \mathbf{c}) = \sum_{i=1}^{N} (x_i - c_i)^2 \tag{5}$$

The distortion (D) is then:

$$D = \mathbf{E}[\|\mathbf{x} - \mathbf{Q}(\mathbf{x})\|^2] = \sum_{i=1}^{K} \int_{I_i} \|\mathbf{x} - \mathbf{Q}(\mathbf{x})\|^2 f_x(\mathbf{x}) d\mathbf{x} \tag{6}$$

The optimal quantizer over a certain input statistic is the one that minimizes (6). Moreover, it must satisfy the following two conditions:

- **Nearest Neighbour condition**: given a codebook, the optimal partition of the input space is the one which gives minimum distortion:

$$I_i = \{\mathbf{x} : \quad d(\mathbf{x}, \mathbf{y}_i) \leq d(\mathbf{x}, \mathbf{y}_j) \quad \forall j \neq i\}, \quad i = 1, 2, ..., K \tag{7}$$

- **Centroid condition**: codevectors of the optimal codebook are the centroids of the decision regions:

$$\mathbf{c}_i = \int_{I_i} \mathbf{x} f_{\mathbf{x}|\mathbf{x} \in I_i}(\mathbf{x}|\mathbf{x} \in I_i) d\mathbf{x}, \quad i = 1, 2, ..., K \tag{8}$$

LBG Algorithm

LBG algorithm is one of the most used method for codebook design and satisfies the optimal conditions. During encoding procedure each image is partitioned in a set of non-overlapping blocks with dimension $n \times n$ pixels. They are reshaped into a vector form and then compressed by the index of the closest vector in the codebook. By building blocks of adjacent pixels we can exploit the correlation that exits among them in an image. An optimal codebook design is essential for reducing the distortion introduced by VQ. This routine is based on Lloyd-Max algorithm but it does not require the statistical knowledge of the source, instead, the quantizer is designed on the basis of the actual data to encode or a representative training set.
In this case, LBG codebook computation starts by collecting blocks represented in vector form over all the training images. Therefore, let's consider a training set of the form:

$$S = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_L\} \tag{9}$$

where each $\mathbf{x}_i$ is an N-dimensional vector and L represents the total number of training instances collected from the images.
Then, the following steps are performed:

1. Compute an initial codebook with one of the techniques described later on this report:

$$\left\{\mathbf{c}_1^{(0)}, \mathbf{c}_2^{(0)}, ..., \mathbf{c}_K^{(0)}\right\} \tag{10}$$

Initialize: $t = 1, \quad D^{(0)} = \infty, \quad \epsilon = 0.01 \quad$ (ϵ can take different values).

2. Compute the set of decision regions as follows:

$$I_i^{(t)} = \left\{\mathbf{x} \in S : \quad \|\mathbf{x} - \mathbf{c}_i^{(t-1)}\|^2 \leq \|\mathbf{x} - \mathbf{c}_j^{(t-1)}\|^2 \quad \forall j \neq i\right\}, \quad i = 1, 2, ..., K \tag{11}$$

3. Compute the new codebook:

$$\mathbf{c}_i^{(t)} = \frac{1}{\left|I_i^{(t)}\right|} \sum_{\mathbf{x} \in I_i^{(t)}} \mathbf{x}, \quad i = 1, 2, ..., K \tag{12}$$

4. Evaluate the distortion at step t:

$$D^{(t)} = \frac{1}{|S|} \sum_{\mathbf{x} \in S} \|\mathbf{x} - \mathbf{Q}(\mathbf{x})\|^2 \tag{13}$$

5. Evaluate terminate condition:

$$\frac{D^{(t-1)} - D^{(t)}}{D^{(t)}} < \epsilon \tag{14}$$

if terminate condition is met then stop, otherwise set $t = t + 1$ and continue from step **2**.

A possible drawback in LBG implementation is that distortion (D) may not be globally minimized, however for sure it does not increase over consecutive iterations. An important factor for obtaining an optimal codebook is its initialization at step (**1**). Several techniques are available for this purpose, among them two well known techniques are: *random* initialization and *splitting*. In the former, codebook initialization is achieved by selecting at random K vectors from the training set S. On the other hand, *splitting* method requires a slightly different approach to the standard algorithm. In fact, it begins by designing a vector quantizer with a codebook of size 1, which is the average over the entire training set vectors. From this output point, in the next iteration, the initial codebook for a two levels quantizer can be doubled by applying (15) to the existing codebook. Once LBG algorithm has converged on this codebook, a four levels quantizer is obtained by applying the same formula to each codeword of the previous quantizer. In this manner, we keep doubling the codebook size until the target size is reached. Then, LBG performs the usual operations until any stop criterion is met.

$$(1+\alpha)\times \mathbf{c}, \quad (1-\alpha)\times \mathbf{c} \quad \forall \mathbf{c} \in \textit{previous codebook} \tag{15}$$

The *empty cell* problem can arise during the execution of LBG algorithm. If a cell is empty (i.e. no vector belonging to S has been assigned during decision regions computation), we substitute the representative for that region with a vector, picked at random, that belongs to the cell with highest cardinality or distortion.

(iii) Simulations setup

All the code for this project has been written in MATLAB.
In this work `imageDatastore` function has been used in order to speed up images retrieval and manipulation. The entire dataset is composed by 90 gray-scale images of size 512×512. The training set consists of the same 26 images while the test set of the same 26 images picked at random from the remaining dataset (i.e. entire dataset without training images), for each type of configuration involved in the study.
In this project the code has been structured in order to maintain flexibility and simplicity, therefore it has been splitted in several basic functions. Among them, `createCodebook` creates a codebook given a set of $n \times n$ blocks collected from each training image. Also, it receives as inputs: a type of codebook initialization technique between *random* and *splitting*, a target size for the codebook and, ϵ which represents the terminate condition. This function encapsulates other routines: `splitCodebook` implements the *splitting* version of LGB by doubling at each iteration the size of the codebook until target size is reached. In addition, the "splitting" is performed using equation (15) with α as tunable parameter; or simply by adding and subtracting a customizable value from each codevector in the current codebook. Inside `createCodebook` the *while loop* that handles codebook creation is iterated until a termination condition (ϵ) is met or a maximum number of iterations (set to 25) is reached.
Given a computed codebook, the function `encodeImage` encodes an image by replacing each vectorized block with its closest codevector in the codebook; `decodeImage` performs the inverse operation.
Source code and the dataset are available at [1].

(iv) Performance Analysis

The vector quantizer, developed with LBG strategy, has been evaluated over two different scenarios. In the first one, blocks of size $L = 2 \times 2 = 4$ at 2 bit/pixel have been taken into account, while in the second scenario 4×4 blocks at 0.5 bit/pixel have been considered. Given a certain codebook size K and block size L the rate is computed according to:

$$R = \frac{\log_2(K)}{L} \tag{16}$$

As a consequence, given the target rate in bit/pixel, the codebook size can be computed as follows:

$$K = 2^{RL} \tag{17}$$

Using the above formula results in a final codebook of size 256 for both scenarios.

Remark: In the following figures, an image surrounded by a cyan-colored contour has been encoded considering blocks of dimension $L = 4 \times 4 = 16$, therefore this contour identifies images belonging to the second scenario, whereas images without that refer to the first scenario. Moreover, in Fig.1a it's depicted the reference image for the training set while in Fig.1b the one for the test set. The goodness of various LBG quantization processes is evaluated in terms of Peak Signal to Noise Ratio (PSNR).

Moreover, in results of Figs.4,5,6, the test images were in number equal to the training images, and they were taken at random from the total test set, as previously stated.

(a) One image from train set

(b) One image from test set

Figure 1: Original train and test images used for comparisons

For each different scenario and initialization technique three values for the terminate condition (ϵ) have been taken into account. From Figs.2,3 and clearly from Figs.4,5,6, we can see that for a decreasing value of the terminate condition ($\epsilon = 0.1 \rightarrow 0.01 \rightarrow 0.001$) the PSNR slightly increases for both *random* and *splitting* initialization methods, and even considering training and test images. This is due to the fact that a smaller ϵ allows for better convergence of the LBG algorithm to a local or global minimum of the distortion function. Anyway the decrease in distortion (i.e. increase in PSNR) is not too evident in both scenarios, this is caused by the fact that the codebook size ($K = 256$) is much smaller than the number of samples (vectors gathered from training images) used during codebook creation in both cases where $L = 4$ and $L = 16$.
In addition, there's no much difference between the two type of initialization techniques: *random* and *splitting*, as we can see from Figs.2a,b,c vs Figs.3a,b,c (for $L = 4$) and Figs.2d,e,f vs Figs.3d,e,f (for $L = 16$) and also in Figs.4,5,6. Hereby, the performance of the two methods are really closed to each other, even if the *splitting* technique should guarantee better convergence, since it expands the codebook over the distribution of vectors in space, and therefore it should assure a better location of the centroids over the input distribution. This negligible performance gap is due to the fact that, again, we are considering codebooks of small size with respect to the total number of vectors.
From Fig.2, we can see that the scenario with $L = 4$ guarantees better performance in terms of PSNR, and therefore lower distortion, than the case $L = 16$. In the latter scenario, considering blocks of such size leads to the introduction of visible artifacts for the encoded image. However, it ensures a four times higher compression ratio than the former case, since we have blocks four times larger in the second scenario and we need in both cases 8 bits for representing the codevector's index of each block. So, the introduction of artifacts is reasonable since we are considering codebooks of the same size but different vector lengths for the two scenarios. Performance of the $L = 16$ case can approach the $L = 4$ case by augmenting the codebook size but this will come at the expense of a lower achievable compression rate, given that we need more bits in order to represent codebook's indexes.
As we can see in Figs.4,5,6, the average PSNR over the test images is smaller than the one over the training images. This result follows the expectation that a codebook obtained over the training images could not guarantee the same performance also over the test set. However this is not completely true, the performance depends on the figures that are considered during LBG algorithm execution (codebook creation). In that sense, if they are representative of all test images, we would have obtained an overall average PSNR much closer to the one for the training set. Therefore in LBG, it's really important the choice of training images as they need to be representative of the total set of images, if possible.

(a) $\epsilon = 0.1$ PSNR = 30.167

(b) $\epsilon = 0.01$ PSNR = 30.565

(c) $\epsilon = 0.001$ PSNR = 30.898

(d) $\epsilon = 0.1$ PSNR = 24.861

(e) $\epsilon = 0.01$ PSNR = 25.041

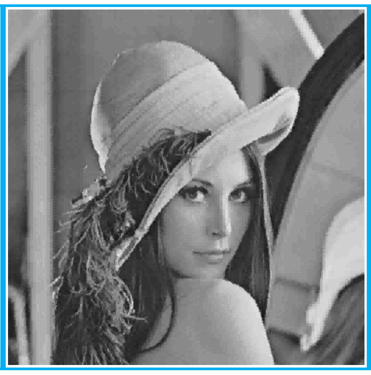
(f) $\epsilon = 0.001$ PSNR = 25.35

(g) $\epsilon = 0.1$ PSNR = 34.899

(h) $\epsilon = 0.01$ PSNR = 35.059

(i) $\epsilon = 0.001$ PSNR = 35.074

(j) $\epsilon = 0.1$ PSNR = 29.86


(k) $\epsilon = 0.01$ PSNR = 30.161

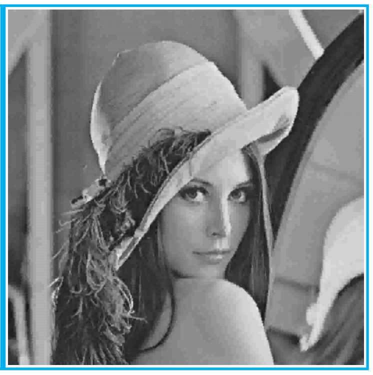

(l) $\epsilon = 0.001$ PSNR = 30.235

Figure 2: Encoding train and test pictures with LBG-random

(a) $\epsilon = 0.1$ PSNR = 30.331

(b) $\epsilon = 0.01$ PSNR = 30.593

(c) $\epsilon = 0.001$ PSNR = 30.824

(d) $\epsilon = 0.1$ PSNR = 24.868

(e) $\epsilon = 0.01$ PSNR = 24.991

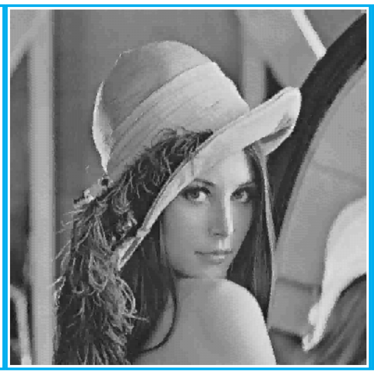
(f) $\epsilon = 0.001$ PSNR = 25.269

(g) $\epsilon = 0.1$ PSNR = 35.228

(h) $\epsilon = 0.01$ PSNR = 35.111

(i) $\epsilon = 0.001$ PSNR = 35.026

(j) $\epsilon = 0.1$ PSNR = 30.134


(k) $\epsilon = 0.01$ PSNR = 30.163

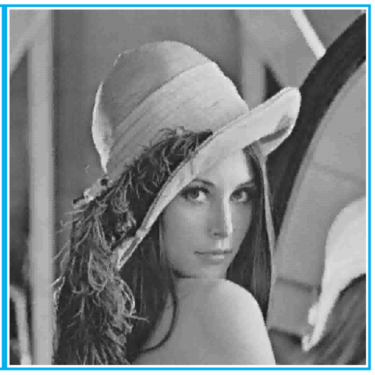

(l) $\epsilon = 0.001$ PSNR = 30.222

Figure 3: Encoding train and test pictures with LBG-splitting

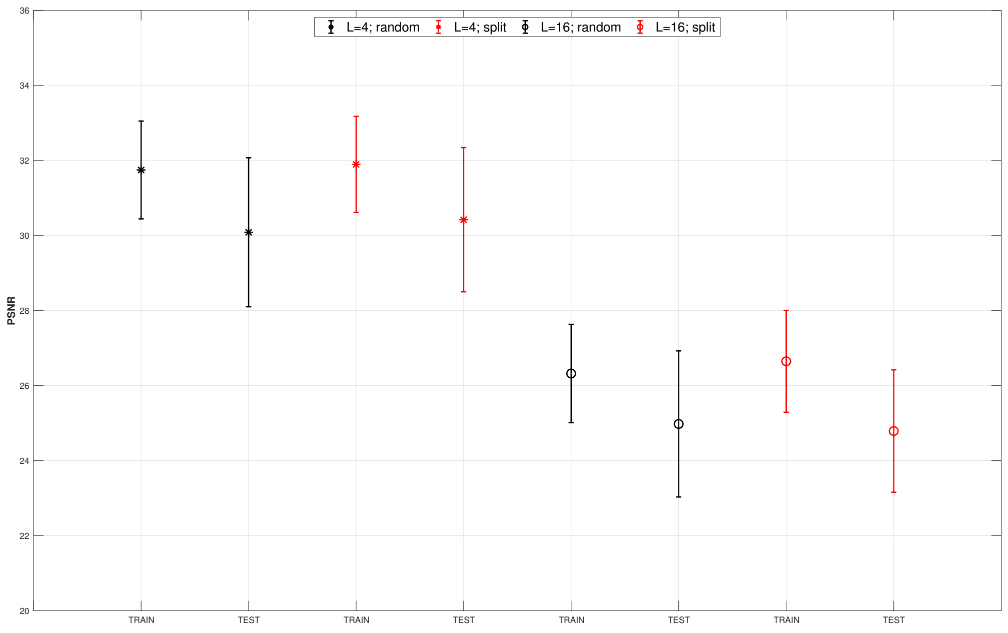


Figure 4: Average PSNR with 95% CI for train and test sets, $\epsilon = 0.1$

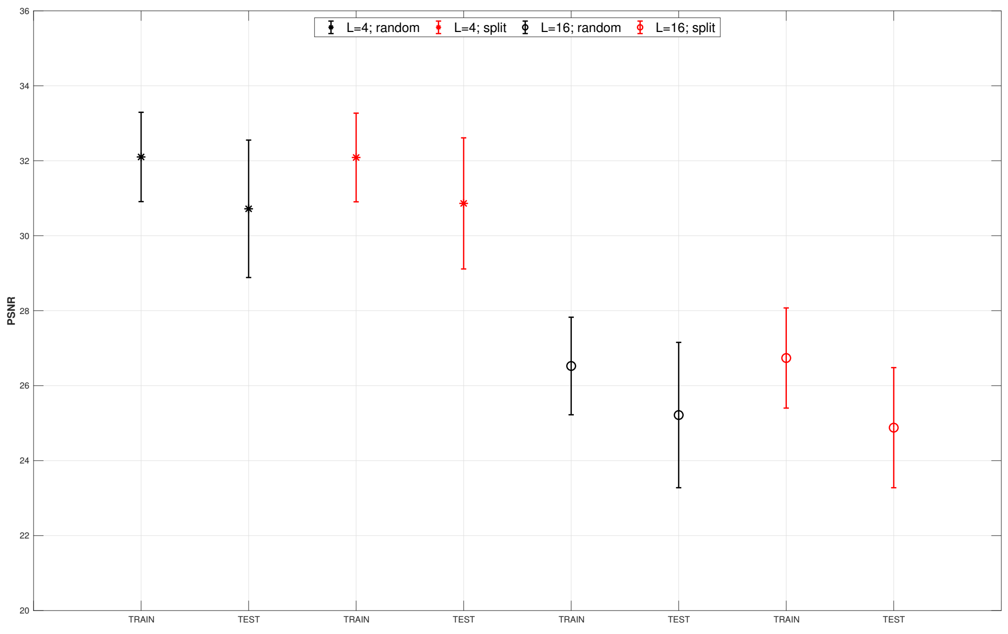


Figure 5: Average PSNR with 95% CI for train and test sets, $\epsilon = 0.01$

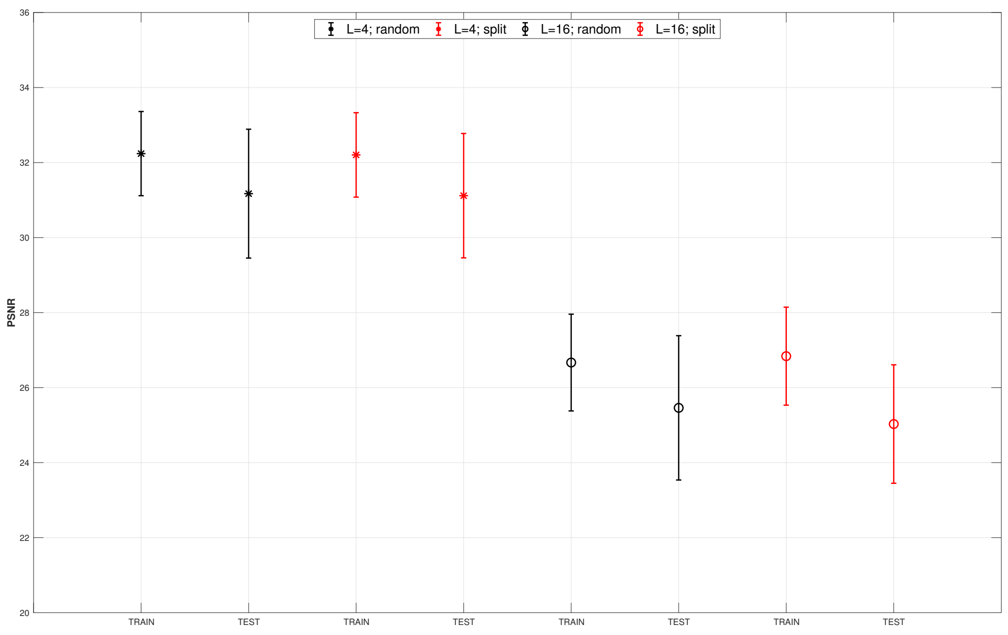


Figure 6: Average PSNR with 95% CI for train and test sets, $\epsilon = 0.001$

Initialization with image's pyramid

In this project, an initialization technique based on building pyramids of training images has been developed and studied. For each training image a Gaussian pyramid as been constructed using `impyramid` Matlab's function. Then for each reduced image, blocks of dimension L have been extracted, thus creating a small set of representative vectors. Finally the initial codebook is obtained by randomly selecting 256 vectors from this set. The objective is to initialize the codebook with the most representative blocks, obtained through pyramidal reduction of each image over the training set. As depicted in Fig.7, this method guarantees almost the same performance against other methods, but at the same time it has shown faster convergence, especially with small values for ϵ.

(v) Conclusions

In this project, the LBG algorithm for vector quantization has been developed and studied. The results has been shown on the basis of different initialization techniques and block dimension L. When dealing with codebooks of small size (especially in the case of blocks with dimension 16 due to the rate constraint of 0.5 bit/pixel), the increase of performance for smaller value of the threshold ϵ is not significant given the much higher convergence time needed. Also the difference exploiting different initialization techniques is not evident in this case. In addition, a small codebook causes a visible introduction of artifacts and imperfections when blocks of dimension $L = 16$ instead of $L = 4$ are considered, however an higher compression rate can be obtained. Therefore, one can conclude that there are many trade-offs that can be considered in the design of the best LBG variant for a given source.
Moreover, in this work, a particular initialization technique based on building several pyramids for each training image has been investigated, and it has demonstrated to reach almost the same level of performance when compared with the standard techniques and a faster convergence time measured in iterations. How-

(a) $\epsilon = 0.1$ PSNR = 30.841

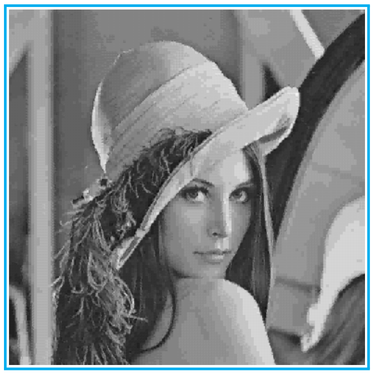
(b) $\epsilon = 0.001$ PSNR = 31.685

(c) $\epsilon = 0.1$ PSNR = 24.665

(d) $\epsilon = 0.001$ PSNR = 25.431

(e) $\epsilon = 0.1$ PSNR = 34.937

(f) $\epsilon = 0.001$ PSNR = 35.283


(g) $\epsilon = 0.1$ PSNR = 29.746

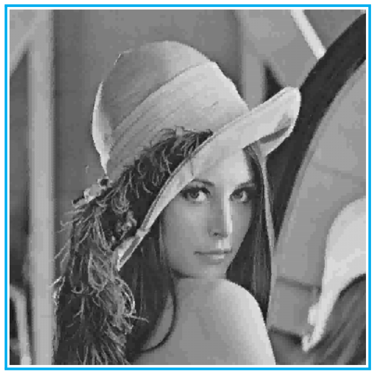

(h) $\epsilon = 0.001$ PSNR = 30.212

Figure 7: Original train and test images using tree initialization

ever, since the codebook size involved during the simulations was small, it could be of interest testing this method considering larger codebook sizes and verify if considerably better results can be achieved in terms of quantization distortion and speed of convergence.

References

[1] Source Code and Dataset