# The DUDE Framework For Continuous Tone Images

Master Thesis in Electrical Engineering

Ignacio Francisco Ramírez Paulino

Thesis Advisor:
Dr. Gadiel Seroussi
Mathematical Sciences Research Institute
United States of America

Academic Advisor:
Prof. María Simón

Instituto de Ingeniería Eléctrica
Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay
November 24, 2005

This thesis was prepared in LaTeXusing fonts from the family
*C*omputer Modern.
The following packages were used: *afterpage*, *amsfonts*, *amsmath*, *amssymb*, *babel*, *calc*, *color*, *fancyhdr*, *graphicx*, *ifthen*, *inputenc*, *makeidx*, *pifont*, *psfrag*, *pstricks*, *rotating*, *subfigure*, *textcomp*, *totpages*, *url* and *xspace*.
All the figures were prepared using Xfig (`http://www.xfig.org`) and Open Office Draw (`http://www.openoffice.org/`)

# Contents

# Preface

The original DUDE was developed jointly by my advisor Gadiel Seroussi and his coleagues at Hewlett-Packard Laboratories, Palo Alto[1] (Marcelo Weinberger, Erik Ordentlich, Tsachy Weissman[2] and Sergio Verdú[3]) as a very general and elegant way to denoise signals of many types and dimensions.

In 2004 I was accepted for an internship at Hewlett-Packard Laboratories, Palo Alto. The primary purpose of this internship was to continue the adaptation of the DUDE algorithm to continuous tone images that Giovanni Motta had started on the northern summer of 2003. Giovanni had already set up the basis for the new framework, and good results were already available in his version. My assignments as an intern were to improve his intial results and to make the overall process more efficient so that the DUDE could be used for real-time image restoration from within popular image processing programs.

The present thesis is a continuation of the work done during the internship and spans the period between August 2004 to August 2005. It addresses issues not solved at the end of the internship, and extends it to other types of noise not originally included (uneven Salt & Pepper, $q$-ary symmetric, Z-Channel), and finally includes improvements in both quality of the results and efficiency.

[1]At the time of writing, Gadiel Seroussi retired from Hewlett-Packard Laboratories and is now an Associate Director of the Mathematical Sciences Research Institute, Berkeley, California 94720, USA

[2]Currently with Stanford University, Stanford, California 94305, USA.

[3]Princeton University, Princeton, New Jersey 08544, USA.

# Abstract

The problem of image denoising is a field of research with more than 50 years of history. It is considered part of the more general problem of image restoration and, ultimately, image processing. As such, it has been addressed traditionally by the signal processing community, starting from the works of Wiener [35] in the late 1940's and Kalman [13]. in 1960.

The Discrete Universal DEnoiser (DUDE) [34] proposes a denoising method which can be applied to any kind of discrete sequences of any dimension, including digital signals, and in particular to digital images. This algorithm has been shown to achieve asymptotically the performance of any fixed sliding-window denoiser for any given sequence corrupted by a memoryless channel, as the length of the sequence approaches infinity.

This work proposes variants to the basic algorithm for its application to continuous tone images, for which the source alphabet is typically very large and the asymptotic properties of the DUDE as originally presented become less relevant. The goal is achieved by exploiting a priori knowledge of the structure of such sequences.

# 1 Introduction

## 1.1 Some history

The problem of image denoising, included in the more general problem of image restoration and ultimately of image processing, is a field of research with more than 50 years of history since the appearance of television which has drawn considerable attention since the advent of digital images in the late 1970's. Today there are thousands of publications on the field, and many practical applications have benefited from their results.

This problem has been addressed traditionally by the signal processing community starting from the works of Wiener [35] (1949) and Kalman [13] (1960), and most of the existing methods to address it are derived from the classical tools of the field of (Digital) Signal Processing. Among these tools are:

- Probability and statistics: Random Procesess, Ergodic Theory, Markov chains, Markov fields, Hidden Markov models.
- Control Theory: Tracking and prediction (Kalman).
- Signal Processing: Digital/Analog Linear Filters, Wiener Filters, Fourier Analysis, Z-Transform.

In the last 10 years many more mathematical tools have been added to the arsenal. Among these are

- Signal Processing: Wavelet/multiresolution analysis
- Statistics: Advanced probability models
- Functional Analysis: Total Variation
- Dynamic Systems: Partial Differential Equations
- Information Theory: Entropy, Minimum Description Lenght, Prediction

## 1.2 Digital Images

The description of the problem of *digital image denoising* begins with the definition of the subject of the problem: *digital images*. A digital image $\mathbf{x}^{m \times n}$ is defined as a two-dimensional array (grid) of $m \in \mathbb{N}$ rows and $n \in \mathbb{N}$ columns, where $\mathbb{N} = \{1, 2, 3, \ldots\}$ is the set of natural numbers. Each position in the array (a sample or “pixel” –for “picture element”–) is referred by a two-dimensional index $i = (i_1, i_2) \in \mathbb{N}^2$ and is denoted as $\mathbf{x}_i$. The color of each pixel is determined by the value at its position in the array. There are three common interpretations of this value:

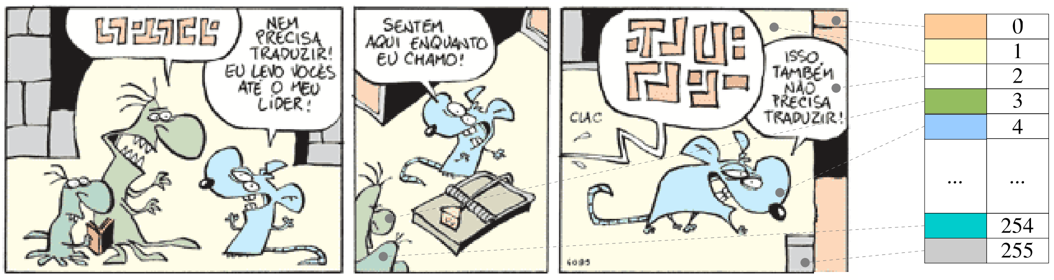

NEM PRECISA TRADUZIR! EU LEVO VOCÊS ATÉ O MEU LÍDER!
SENTEM AQUI ENQUANTO EU CHAMO!
CLAC
ISSO, TAMBÉM NÃO PRECISA TRADUZIR!
0
1
2
3
4
...
...
254
255

Figure 1.1: "Niquel Nausea", an indexed image with a 4-bit (16 colors) palette. Each color in the comic (left) corresponds to a 4-bit index to a position in the palette (right).

- As an index into a palette of colors. These are *indexed images* (Figure 1.1).[1]
- As a light intensity measure of a monochromatic light. Images of this type are called *continuous-tone images* (Figure 1.2).
- As a vector of light intensities in $n$ color bands, usually: red, green and blue (RGB). Images of this type are called *truecolor*. These images can always be decomposed into $n$ monochromatic continuous-tone images, one for each band (Figure 1.3).

This work is restricted to the second case, since its analysis is simpler than the truecolor case, and truecolor images can always be treated as an n-uple of continuous-tone images.[2]

Computers store numerical values with finite precision. The light intensity at each pixel is no exception and it will have to assume one of a finite set of values $\mathcal{A} = \{0, 1, \ldots, M-1\}$, where 0 represents the minimum intensity (black), $M-1$ is the maximum intensity (white), and the symbols between 0 and $M-1$ represent continuously increasing intensities from black to white. The set $\mathcal{A}$ is called an *alphabet*, and its size $|\mathcal{A}| = M$ defines the precision available to represent the different intensities. This size is determined by the number of bits-per-pixel (bpp) as $M = 2^{bpp}$. A typical continuous-tone image has 8 bpp, which yields $M = 2^8 = 256$ possible intensities. Such is the case of the images studied in this work.[3]

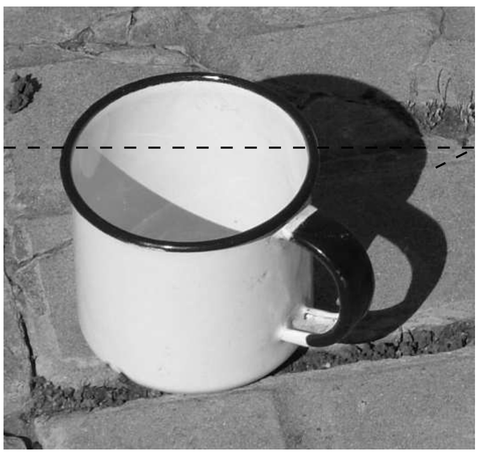
## 1.3 The problem of image denoising

Digital images such as digital photographs or scanned documents are subjected to a series of phenomena that result in some or all of their pixels being modified in undesired ways (corrupted). An example of this problem is the *thermal noise* at the CCD (Charge-Coupled Devices) arrays which sense the incoming light in most digital cameras. In these devices, the intensity of each pixel is proportional to the number of photons that hit each cell in the CCD array. The thermal noise is produced by photons arriving from nearby atoms and is an effect which happens at any temperature above absolute zero, increasing proportionally to the temperature of the device. Figure 1.4 shows a scheme of this process.

[1]Image taken from `http://niquelnausea.terra.com.br` as of August 2005

[2]This does not mean that working with all bands at once is equivalent. Algorithms based directly on color can exploit the fact that bands are not independent of each other.

[3]There is nothing special about this value, however, most of the techniques described would apply to other values of bpp.


(a) Continuous-tone image. Row 130 is shown dashed.

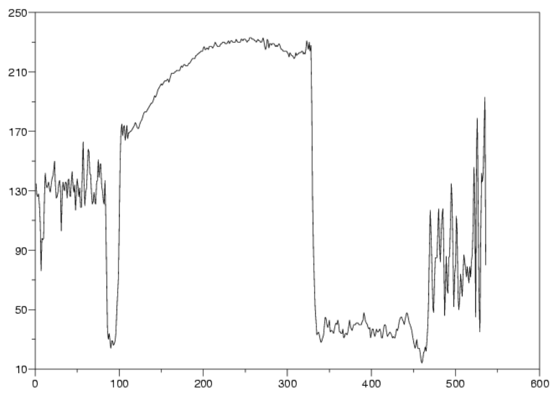

(b) Pixel value (intensity) graph for row 130.

Figure 1.2: "Coffee Cup", a continuous tone image of 512 × 512 pixels. The graph to the right corresponds to the pixel values in row 130 starting from the upper row (at about 1/3 of its height).

Another typical case is the degradation of negative film by dust, scratches or fungi, which introduce tiny specks that can be very notorious, degrade the aesthetics of the image or hide vital parts of it. This type of noise also appears in some faulty digital cameras where some of the pixels in the sensing device are defective. This noise is usually called "dust and scratches" or "salt and pepper" for its visual effect.

The effect of both types of noise can be seen in Figure 1.5 for the "Coffee Cup" image.

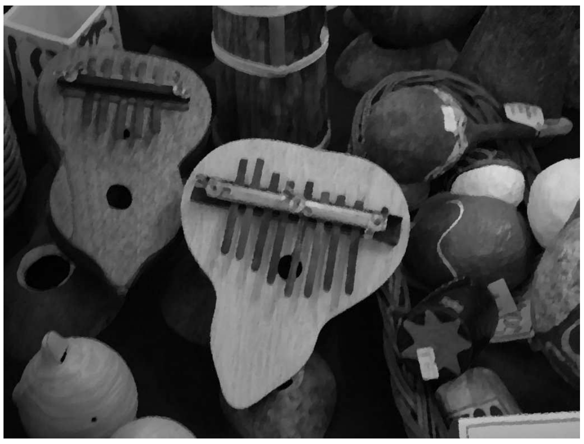
*The problem of image denoising is to correct or guess those faulty or deleted pixels so that the resulting image is closer to the original image. Ideally, the result should be more inteligible, and/or more pleasant to the human eye than the observed noisy image.*

The general process of image degradation can be described as in the diagram of Figure 1.6. In this diagram, the noisy image $\mathbf{z}^{m\times n}$ is the result of the clean (unknown) image $\mathbf{x}^{m\times n}$ after going through a *transmission channel*. Throughouth the rest of this document, $\mathbf{x}^{m\times n}$ will be used to refer to the clean image and $\mathbf{z}^{m\times n}$ to the noisy image.

The transmission is carried out sample by sample (for example from top to bottom and from left to right), and the channel substitutes each clean sample $\mathbf{x}_i$ with a noisy sample $\mathbf{z}_i$ with a given probability $P(Z = \mathbf{z}_i | X = \mathbf{x}_i)$. The channels considered in this work are *discrete memoryless channels* (DMC). They are *memoryless* because the probability of $\mathbf{z}_i$ depends only on the value of the $\mathbf{x}_i$ and is independent of the noisy value at any other position in the image. They are also discrete, as the as the alphabet of the input and output images (normally the same for digital images) is discrete. A DMC is characterized by its *transition matrix* $\Pi = ((\pi_{ij}))_{i,j\in\mathcal{A}}$, where each element $\pi_{i,j} = P(Z = j|X = i)$ is the probability that the channel outputs a noisy sample with value $j$ when the clean (unknown) sample value was $i$.

(a) Color image.

(b) Red component/band.


(c) Green component/band.

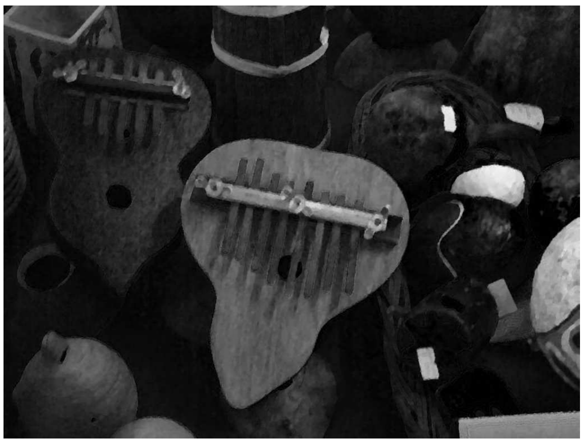

(d) Blue component/band.

Figure 1.3: "Kalimbas", a truecolor RGB image. Each band is represented as an 8-bit continuous tone image.

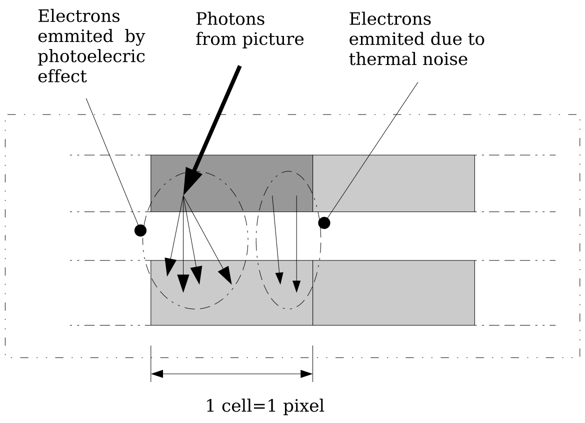
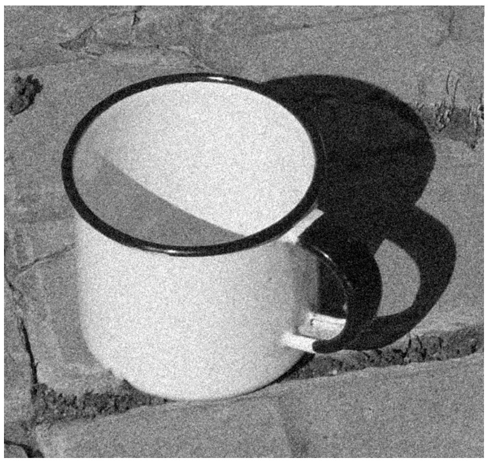
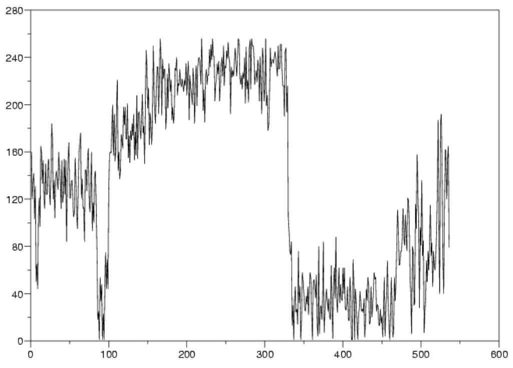

Electrons emmited by photoelecric effect
Photons from picture
Electrons emmited due to thermal noise
1 cell=1 pixel

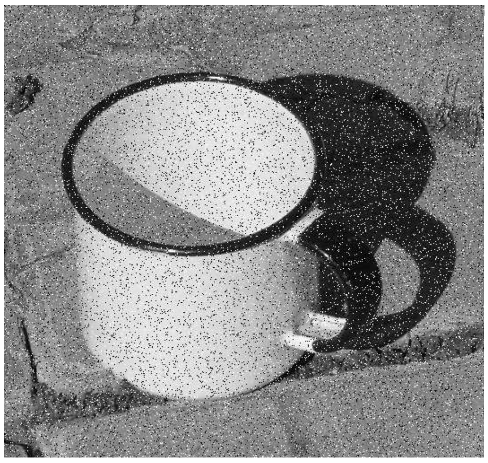
Figure 1.4: Scheme of thermal noise in CCD arrays.



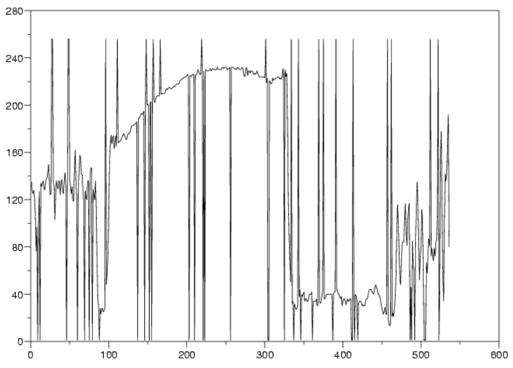
(a) Coffee cup currupted by thermal noise.



(b) Intensity curve of row 130 (about 1/3 of height from the top).



(c) Coffee cup currupted by dust and scratches.



(d) Intensity curve of row 130.

Figure 1.5: Examples of “Coffee Cup” corrupted by different types of noise.

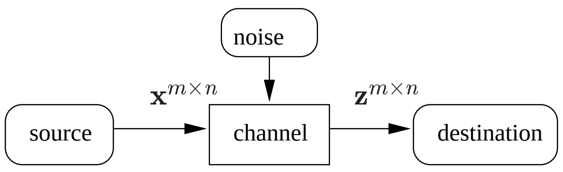


Figure 1.6: Theoretical scheme of the image degradation process.

**Example** The *Binary Simmetric Channel* (BSC) operates with $\mathcal{A} = \{0,1\}$ as the input and output alphabet. The channel inverts the value of the each input symbol $\mathbf{x}_i$ with probability $p$, and leaves it untouched with probability $1-p$. Thus for each input symbol $\mathbf{x}$,

$$\begin{aligned} P(Z=1|x=0) &= P(Z=0|x=1) = p \\ P(Z=0|x=0) &= P(Z=1|x=1) = 1-p \end{aligned}$$

and the channel transition matrix $\Pi$ is

$$\Pi = \begin{bmatrix} 1-p & p \\ p & 1-p \end{bmatrix}$$

.

**Example** The Z-Channel. Here again the input and output alphabets are $\mathcal{A} = \{0,1\}$. In this case a clean symbol with value 0 has a probability $p$ of being substituted by a 1, and probability $1-p$ of going through the channel untouched. However, the symbol 1 is always kept untouched. This results in the following channel transition matrix:

$$\Pi = \begin{bmatrix} 1-p & p \\ 0 & 1 \end{bmatrix}$$

## 1.4 Notation

This section formalizes the notation to be used throughout the rest of the document. Some of it has already been introduced previously in this chapter, and is repeated here to provide a reference.

Concepts that appear for the first time are shown in *italic* text. Text that appears in `typewriter` font denotes a *configurable parameter* of an algorithm, for example `cond_tex_bits`.

Sets of numbers are represented by letters such as $\mathbb{N}$, $\mathbb{Z}$ or $\mathbb{R}$. The set of integers is $\mathbb{Z} = \{\ldots,-2,-1,0,1,2,\ldots\}$, the set of naturals (strictly positive integers) $\mathbb{N} = \{1,2,3,\ldots\}$, and $\mathbb{R}$ represents the set of real numbers.

Indexed arrays (or vectors) are enclosed in parenthesis. Example $\mathbf{v} = (1,0,0)$. When specified, the indexing domain is specificed as a subscript expression: $(h_i)_{1\leq i\leq k}$. For multidimensional arrays a similar notation with a number of parenthesis corresponding to the dimension of the array is used. For instance, matrices are denoted as $\Pi = ((\pi_{ij}))_{i,j\in\mathcal{A}}$, where $\pi_{ij}$ is the element at position $i,j$. When appropiate, the alternate notation $\pi(i,j)$ is used to refer to such element .

Set definitions are enclosed in $\{\}$. Example $\mathcal{A} = \{\text{cloudy}, \text{sunny}, \text{rainy}\}$.

An image of size $m\times n$ is shown in bold face with its dimensions specified as a superscript, as in $\mathbf{x}^{m\times n}$. To simplify notation, the concept of multidimensional indexes $i = (i_1, i_2)$ is used to refer to a particular symbol in the image, such as $\mathbf{x}_i$. As images are represented as arrays of size $m\times n$, $\mathbf{x}^{m\times n}$ is equivalent to $((\mathbf{x}))_{1\leq i_1\leq m, 1\leq i_2\leq n}$ and $\mathbf{x}_i$ is equivalent to $\mathbf{x}(i_1,i_2)$.

## 1.5 Document organization

After this brief introduction to the problem of image denoising, Chapter 2 presents the various types of noise studied in the denoising literature. Chapter 3 gives a short review of image existing denoising algorithms. Chapter 4 describes the basic DUDE algorithm and its problems. Chapter 5 presents the general tools used to address these problems and Chapter 6 follows by describing the resulting proposed solution. Chapter 7 shows the results that were obtained, Chapter 8 give the conclusions obtained from the former results, and finally the future lines of research are outlined in Chapter 9.

# 2 Noise models for digital images

## 2.1 Additive noise

### 2.1.1 The Gaussian channel

The discrete Gaussian channel is modeled after the *continuous additive white gaussian noise model*. The latter is of special importance to many real life problems since it models many natural processes, such as transmission over analog channels [4, pp. 239-265], and has been extensively studied since the beginning of the field of signal denoising [13]. The discrete gaussian channel serves as a model for the effect of the continuous channel on the physical aspects of signal level discretization present in digital acquisition devices. Such is the case of the thermal noise in CCD devices described in Chapter 1. Since the advent of digital images, this channel has also become a classical model for image degradation, and many of the algorithms studied in this work (to be described in Chapter 3) are designed specifically to attack this type of noise.

The additiveness of the channel means that each corrupted pixel $\mathbf{z}_i$ is the result of the addition of the clean (unknown sample) $\mathbf{x}_i$ and a random noise sample $\mathbf{n}_i$. The noise is *white* when its samples are statistically independent of each other, and their mean value is zero. Finally, the channel is Gaussian because the probabilities of the noise sample values obey a *Normal continuous distribution* $N_{\mu,\sigma}$ of mean $\mu$ and variance $\sigma$,

$$p(n) = N_{\mu,\sigma}(n) = \frac{1}{\sqrt{2\pi}\sigma} \exp^{\frac{(n-\mu)^2}{\sigma^2}}$$

The white nature of the noise implies $\mu = 0$ so that $p(n) = N_{0,\sigma}$. As the channel is additive, the random variable modeling a (continuous) noisy sample $Z$ is related to its corresponding discrete (non random) clean sample $\mathbf{x}$ and the random variable for the noise sample $N$ by equation (2.1)

$$Z = N + x, x \in \mathcal{A} \tag{2.1}$$

Using (2.1), the resulting (continuous) probability density function of $Z$ conditioned on $x = a$ is

$$p(Z|x = a) = N_{a,\sigma}$$

As the channel is discrete, the continuous value of $Z$ has to be mapped to return to the original discrete alphabet. This model assumes that the value of the continuous random variable is rounded to the nearest integer value in the alphabet,

$$Z = \begin{cases} 0 & , Z \leq 0 \\ M-1 & , Z \geq M-1 \\ round(Z) & , \text{otherwise} \end{cases}$$

Defining the round($\cdot$) operation as

$$\text{round} : \mathbb{R} \to \mathbb{Z}, \text{round}(y) = j \in \mathbb{Z}, j - 0.5 \leq y < j + 0.5$$

the elements of the channel transition matrix are obtained as

$$\pi_{ij} = P(Z = j | x = i) = \begin{cases} P(-\infty < Z < 0.5) & , j = 0 \\ P(M - 1 - 0.5 < Z < +\infty) & , j = M - 1 \\ P(j - 0.5 \leq Z < j + 0.5) & , 0 < j < M - 1 \end{cases}$$

Using (2.1), $N = Z - x$ and

$$P(Z = 0 | X = i) = P(-\infty < N < -i + 0.5) \quad (2.2)$$
$$P(Z = M - 1 | X = i) = P(M - 1 - i - 0.5 \leq N < +\infty) \quad (2.3)$$
$$P(Z = j | X = i) = P(j - i - 0.5 \leq N < j - i + 0.5), 1 < j < M - 1 \quad (2.4)$$

These probabilities are obtained by integrating the continuous normal density function of the noise over the specified interval,

$$P(z < \alpha) = \int_{y=-\infty}^{y=\alpha} N_{0,\sigma}(y)$$

## 2.2 Non-additive noise

### 2.2.1 The erasure channel

The noise models described from subsections 2.2.2 through 2.2.5 are non-additive, meaning that the random variable modeling the noisy samples $Z$ is not related to the r.v. modeling the clean sample $X$ through an operation involving the addition of an indenpendent noise variable $N$. Note that thelatter definition of non-additivity includes any relationship that is not a sum (for example *multiplicative noise* where the relationship could be $Z = N * X$).

Here, The discussion will be focused on the cases where each noisy sample $\mathbf{z}_i$ is either equal to $\mathbf{x}_i$ or is replaced by an *erasure* value which has *no relationship* with the value of $\mathbf{x}_i$. One channel commonly used as an example of this behavior is the *Erasure Channel* [4, pp. 187-189]. Although it is not studied in this work as a channel by itself, it captures the main properties that are common to the non-additive channels presented here.

Given an input alphabet $\mathcal{A} = \{\alpha_1, \alpha_2, \ldots, \alpha_{M-1}\}$, $|\mathcal{A}| = M$, the Erasure Channel substitutes each input symbol $\mathbf{x}_i$ by an *erasure symbol* $e \notin \mathcal{A}$, regardless of the value of $\mathbf{x}_i$ with probability $P_e$, or leaves it untouched so that $\mathbf{z}_i = \mathbf{x}_i$. Thus the output alphabet of the Erasure Channel is $\mathcal{A}^* = \{\alpha_1, \alpha_2, \ldots, \alpha_{M-1}, \mathbf{e}\}$, $|\mathcal{A}| = M + 1$ and the resulting transition matrix is

$$\Pi = \begin{bmatrix} 1 - P_e & 0 & \cdots & 0 & P_e \\ 0 & 1 - P_e & \cdots & 0 & P_e \\ \vdots & \vdots & \ddots & 0 & \vdots \\ 0 & 0 & \cdots & 1 - P_e & P_e \end{bmatrix}$$

Because the event $Z = e$ does not depend on $X$, a noisy symbol $\mathbf{z}_i = e$ does not contain any information about $\mathbf{x}_i$. This is an important difference with respect to additive channels such as the Gaussian Channel and has many practical implications.

### 2.2.2 Impulse (Salt & Pepper) channel

In the Impulse Channel – often named "Salt and Pepper" after its visual effect –, each pixel of the image is randomly replaced by either the maximum symbol in the alphabet (*salt*), or the minimum (*pepper*), with a total probability of error $\lambda$ which is evenly distributed among the two cases (i.e. $\lambda/2$ for each of the two possible corrupted symbols); and it is left untouched with probability $1 - \lambda$. The channel transition matrix for this case is

$$\Pi = \begin{bmatrix} 1-\lambda/2 & 0 & \cdots & 0 & \lambda/2 \\ \lambda/2 & 1-\lambda & 0 & 0 & \vdots \\ \vdots & 0 & \ddots & 0 & \lambda/2 \\ \lambda/2 & 0 & 0 & 1-\lambda & \lambda/2 \\ \lambda/2 & 0 & \cdots & 0 & 1-\lambda/2 \end{bmatrix} \tag{2.5}$$

It is useful to view this channel as a variant of an *erasure channel*, where the "erasures" are symbols from the clean sequence alphabet. Being erasures, the noisy samples do not provide any information on the corresponding clean samples.

### 2.2.3 Asymmetric impulse channel

This channel is a simple extension of the Salt & Pepper Channel in which $P(Z = \textit{salt}) = \lambda_s$ and $P(Z = \textit{pepper}) = \lambda_p$ are not equal. The total probability of error is redefined as $\lambda = \lambda_s + \lambda_p$. In this case, the transition matrix is

$$\Pi = \begin{bmatrix} 1-\lambda_s & 0 & \cdots & 0 & \lambda_s \\ \lambda_p & 1-\lambda & 0 & 0 & \vdots \\ \vdots & 0 & \ddots & 0 & \lambda_s \\ \lambda_p & 0 & 0 & 1-\lambda & \lambda_s \\ \lambda_p & 0 & \cdots & 0 & 1-\lambda_p \end{bmatrix} \tag{2.6}$$

### 2.2.4 The Z Channel

This is a special case of the asymmetric impulse channel in which $\lambda_p = 0$ and $\lambda_s = \lambda$, and thus its treatment is the same as the latter. Despite this, the *Z channel* is one of the classical channel models used in information theory and thus it is worth including it as a case of study by itself.

### 2.2.5 The $q$-ary symmetric channel

This is a type of non-additive channel where the total probability of error $\lambda$ is distributed evenly among the noisy symbols. For an alphabet of size $M$ and a clean symbol $x$, the channel will substitute the latter with a noisy symbol $z \neq x$ with probability $\lambda_M = \frac{\lambda}{M-1}$ or leave it untouched with probability $1 - \lambda$. This results in the following matrix

$$\Pi = \begin{bmatrix} 1-\lambda & \lambda_M & \cdots & \lambda_M & \lambda_M \\ \lambda_M & 1-\lambda & \lambda_M & \lambda_M & \vdots \\ \vdots & \lambda_M & \ddots & \lambda_M & \lambda_M \\ \lambda_M & \lambda_M & \lambda_M & 1-\lambda & \lambda_M \\ \lambda_M & \lambda_M & \cdots & \lambda_M & 1-\lambda \end{bmatrix} \tag{2.7}$$

As will be seen later, it presents some additional challenges since its "erasure" nature is less evident than the Impulse Channel and its variants.

## 2.3 Noise measures

When comparing different denoising methods one must define some criterion of what is considered to be a good result. As the ultimate goal is to produce an image that looks "better" to the human eye, the best possible criterion is certainly subjective. However, the problem of finding an objective criterion which approximates the best subjective criterion is a very difficult one. Because of this, and because they are of general use in other problems of image and signal processing, a few objective performance measurements are generally used: *MSE/SNR* and *PSNR*. The MSE (*Mean Square Error)*, is defined as follows:

$$\mathrm{MSE}(\mathbf{z}^{m\times n}) = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}_i - \mathbf{z}_i)^2. \tag{2.8}$$

The SNR (*Signal to Noise Ratio*) measures the relation between the power of the "signal" (the clean image) and the power of the "noise" (which is the MSE).

$$\mathrm{SNR}(\mathbf{z}^{m\times n}) = 10\log\left(\frac{\sum_{i=1}^{N}(\mathbf{x}_i - \mathbf{z}_i)^2}{\sum_{i=1}^{N}\mathbf{x}_i^2}\right) \tag{2.9}$$

Finally, the PSNR (*Peak Signal to Noise Ratio*) is equivalent to the MSE, expressing it in relative logarithmic units (dB) with respect to the "peak" power of the signal. For an 8-bit image, this is

$$\mathrm{PSNR}(\mathbf{z}^{m\times n}) = 10\log\left(\frac{\mathrm{MSE}(\mathbf{z}^{m\times n})}{255^2}\right) \tag{2.10}$$

All those measures give more weight to bigger differences than to smaller ones due to their quadratic nature. This is usually considered to be akin to the subjective perception of noise. Of the three, the PSNR and MSE are the most popular as they do not depend on the power of the image to be denoised and thus they can be averaged throughout a set of test images to produce an "average performance measure" for the test suite. Of them, the PSNR will be the preferred one as it is the most common of the three.

# 3 A review of image denoising

The problem of image denoising has been given an extensive treatment in the literature which makes it impossible to include a comprehensive set of references in this document. Therefore, the discussion will be restricted to some of the more representative denoising algorithms: the classical ones described in text books such as [9] or [10], and the ones which are considered the current state-of-the-art.

An *image filter* is *any* algorithm which takes some image as input and produces an output image as a result. A *denoising filter* is a filter that, given a noisy input image $\mathbf{z}^{m\times n}$, produces an output $\hat{\mathbf{x}}^{m\times n}$ that is *closer* to the unknown clean image $\mathbf{x}^{m\times n}$ that was fed to the transmission channel.

First, the filter techniques which form the basis for most of the common filters found in the literature are presented. Then follows a description of specific filters designed to attack each type of noise.

## 3.1 Neighborhood and window filters

The principle of these filters is to infer the clean pixel $\mathbf{x}_i$ based on the information provided by some pixels on the noisy image located in a neighborhood of its position $i$. Let $W = (i_r)_{1\leq r\leq K}$ be a vector of indexes which are "near" $i$ under some criterion. $W$ is a window index vector and use $\mathbf{x}(W)$ to denote the vector of the *values* of the pixels at the locations specified in $W$, i.e., $\mathbf{x}(W) = (x_{i_1}, x_{i_1}, \ldots, x_{i_K})$. If $W$ includes $i$, $\mathbf{x}(W)$ is called a *window* and if not, it is a *neighborhood* or *context*.

These filters exploit the common assumption that pixels which are close to each other tend to have similar values (for example in smooth regions of the image). In principle, the number and relative location of the pixels which are used by the algorithm can vary for each location $i$.

## 3.2 Sliding-window filters

This is a common case in which the window/neighborhood *shape* is fixed for every index $i$, and its position is centered at $i$. This shape is defined by a *window template*, which is a vector of offsets $T = (d_r)_{1\leq r\leq K}$ of *index offsets* $d_r \in \mathbb{Z}^2$. For a given window template and the position $i$, the corresponding window index vector $W_{i,T}$ is obtained by adding $i$ to each index offset in the template

$$W_{i,T} = (i + d_1, i + d_2, \ldots, i + d_K) \tag{3.1}$$

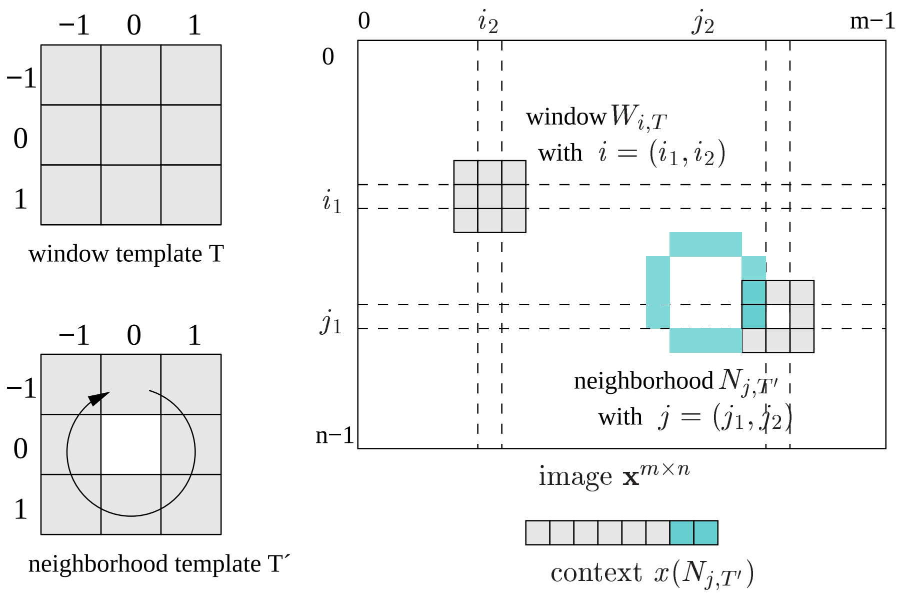


Figure 3.1: A window template $T = \{(0,0),(-1,0),(-1,1),\dots,(0,-1),(-1,-1)\}$, a neighborhood template $T' = \{(-1,0),(-1,1),\dots,(0,-1),(-1,-1)\}$, an image $\mathbf{x}^{m\times n}$, a window, a neighborhood and a context.

The window at position $i$ is obtained as $\mathbf{x}(W_{i,T})$, provided with some convention for the values of the pixels outside the image range $R_{m\times n} = \left\{i = (i_1, i_2) \in \mathbb{N}^2 : i_1 \le m, i_2 \le n\right\}$ (for example, repeating the value of the closest border pixel).

Finally, a *sliding-neighborhood filter* is the case when the template does not include the center, i.e., the offset $(0,0)$. These concepts are depicted in Figure 3.1.

## 3.3 Linear (convolution) filters

These are a special case of the sliding-window filters where the estimated value of the center pixel is a linear function of the window samples. If for each $i$, $j_r$ denotes the $r$-th element of $W_{i,T}$:

$$\hat{\mathbf{x}}_i = \sum_{r=1}^{K} h_r \mathbf{z}_{j_r} \tag{3.2}$$

where $h_r \in \mathbb{R}$ are coefficients assigned to each position (offset) $j_r$ and independent $i$. These filters are also called *convolution* or FIR (Finite Impulse Response) filters, as (3.2) can always be written as a linear convolution

$$\hat{\mathbf{x}}_i = \sum_{k_1=-L_1}^{U_1} \sum_{k_2=-L_2}^{U_2} \hat{h}_k \mathbf{z}_{i-k} \ , \ k = (k_1, k_2) \tag{3.3}$$

where $k = (k_1, k_2)$ are index offsets covering the smallest rectangular region that contains the window template $T$

$$\{(k_1, k_2) : -L_1 \le k_1 \le U_1, -L_2 \le k_2 \le U_2\}$$

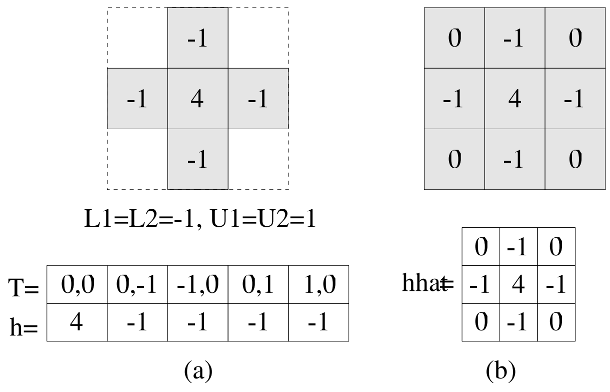


Figure 3.2: From templates and windows to linear convolution kernels. The example here corresponds to the Laplacian operator used to detect borders in images.

and $\hat{h}_k = h_r$ if $T$ contains the offset $k$ at position $r$ or 0 otherwise (see Figure 3.2 for a graphical explanation).

The rectangular 2D array $(\hat{h}_k)_{-L_1<k_1<U_1,-L_2<k_2<U_2}$ constitutes the *linear convolution kernel*. It is also called the *impulse response* of the filter as it coincides with the output of an impulse signal (image) $\delta^{m\times n}$, ($\delta(0,0)=1$ and 0 everywhere else) when the filter is applied to it (trivial by substituting $\delta^{m\times n}$ in equation (3.2)).

These filters are at the core of classical digital signal processing. See [20] for more details on the theory and application of these filters.

## 3.4 Frequency domain filters

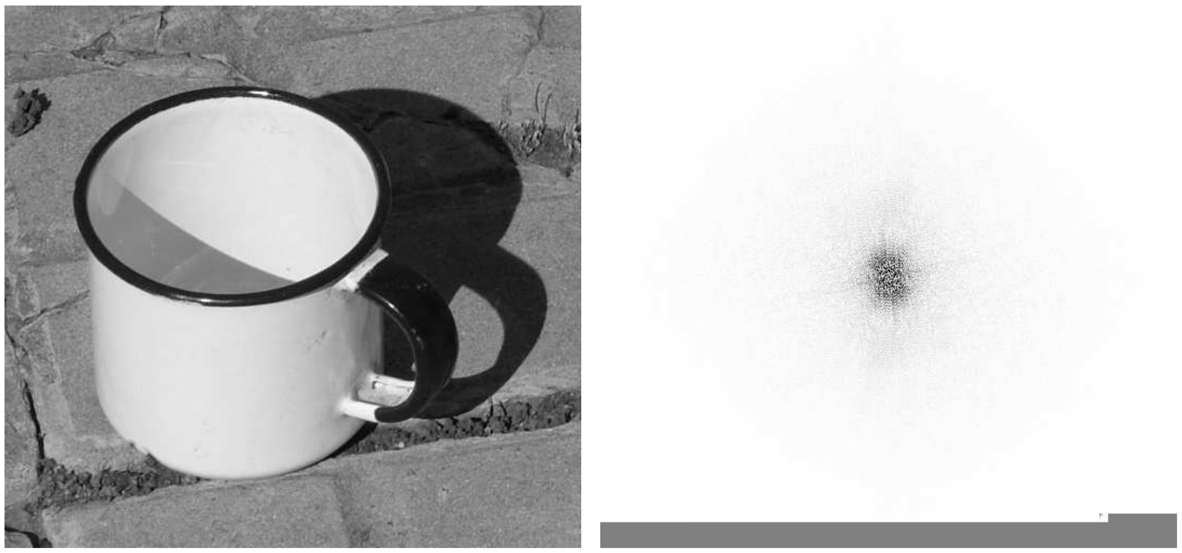
One classical tool for signal processing in general, and for digital images in particular, is the *frequency domain analysis* or *Fourier analysis* (see [20] for a review). It consists of decomposing the image into a set of sine waves

$$\mathbf{x}_{(i_1,i_2)} = \sum_{u=1}^{M}\sum_{v=1}^{N} \mathbf{x}_{(u,v)} e^{j\frac{2\pi u i_1}{M}} e^{j\frac{2\pi v i_2}{N}} \tag{3.4}$$

where $e^{jx}$ denotes *complex exponentiation* and each term $\mathbf{x}_{(u,v)}$ is the $(u,v)$ term of the Fourier Transform $\mathbf{x}^{m\times n} = \mathcal{F}(\mathbf{x}^{m\times n})$, computed as

$$\mathbf{x}_{(u,v)} = \frac{1}{MN}\sum_{i_1=1}^{M}\sum_{i_2=1}^{N} \mathbf{x}_{(i_1,i_2)} e^{-j\frac{2\pi u i_1}{M}} e^{-j\frac{2\pi v i_2}{N}} \tag{3.5}$$

Each coefficient $X_{(u,v)}$ of the Fourier Transform represents the power of the image at the discrete spatial frequency $(2\pi u/M, 2\pi v/N)$. Figure Figure 3.3 shows an image and its Fourier Transform (the Fourier Transform is usually displayed shifted so that the center pixel represents $X(0,0)$, — called the "DC" term as its value is the average of $\mathbf{x}^{m\times n}$).



(a) Continuous-tone image. (b) And its Fourier transform.

Figure 3.3: Fourier transform of an image. The "DC" component is at the center of the DFT.

The *frequency domain filters* use this representation of the image to try to sepparate the noise from the clean image. One example is the family of *lowpass filters*, which assumes that the noise is white and additive of mean 0. In this case, the power of the noise is spread evenly among all frequencies in the Fourier Transform of the noisy image. The lowpass filters assume that the clean image information is concentrated in the lower frequencies and thus the denoising process reduces to removing the higher frequency components of the Fourier Transform while keeping the lower frequencies intact. The many different variants of lowpass filters (see [20] for some of them) differ in the way they define the transition from "low" to "high" frequencies. For instance, a simple "cutoff" filter is defined as

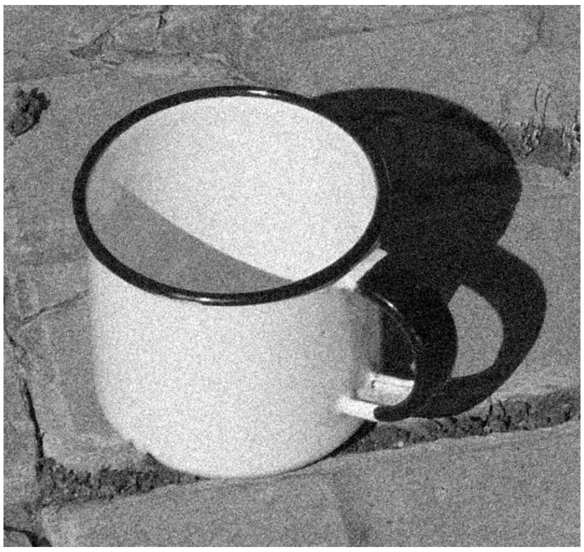
$$\hat{\mathbf{X}}_{(u,v)} = \begin{cases} \mathbf{Z}_{(u,v)} & , \quad \sqrt{u^2+v^2} < f_c \\ 0 & , \quad \text{otherwise} \end{cases}$$

where $f_c$ is the *cutoff frequency.* An example of this filtering technique is shown in Figure 3.4. This filter has a number of problems related to the sharp fall between the "bandpass" region and the "bandstop" region. An inspection of Figure 3.4 shows this effect, known as "ripples", "bandings", or *Gibbs oscilations.* Please refer to signal processing books such as [20] for a theoretical explanation.

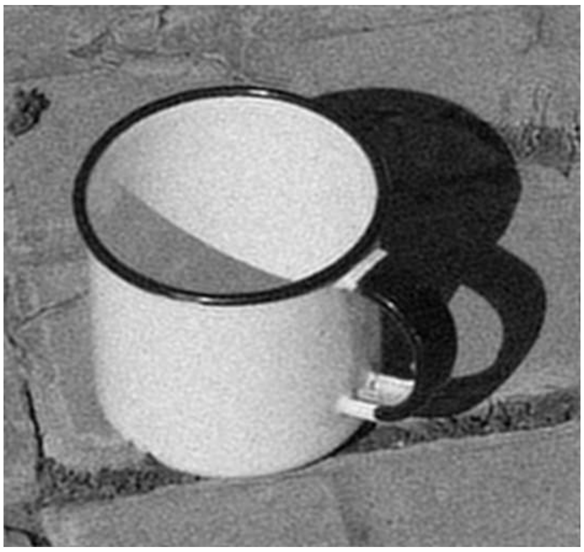
Note that frequency domain filters can be implemented in a perfect or approximate way as linear convolution filters of the type described in the previous section (see [20] for a general method). Furthermore, every linear filter has an associated *frequency response* defined as the Fourier Transform of its impulse response,

$$\mathbf{H}_{(u,v)} = \frac{1}{MN} \sum_{i_1=1}^{M} \sum_{i_2=1}^{N} \mathbf{h}_{(i_1,i_2)} e^{-j\frac{2\pi u i_1}{M}} e^{-j\frac{2\pi v i_2}{N}} \tag{3.6}$$

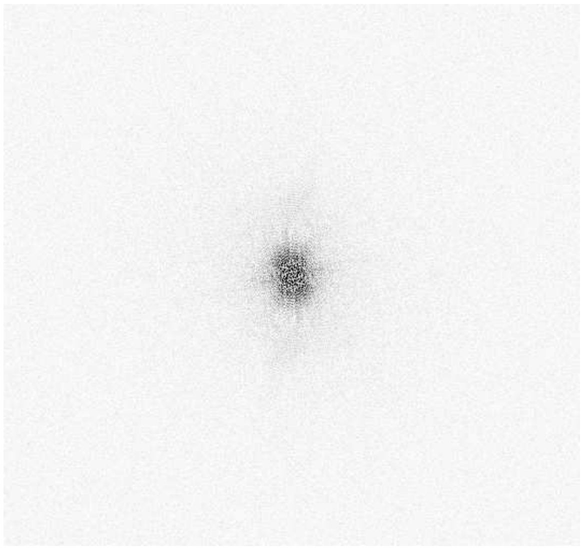
.



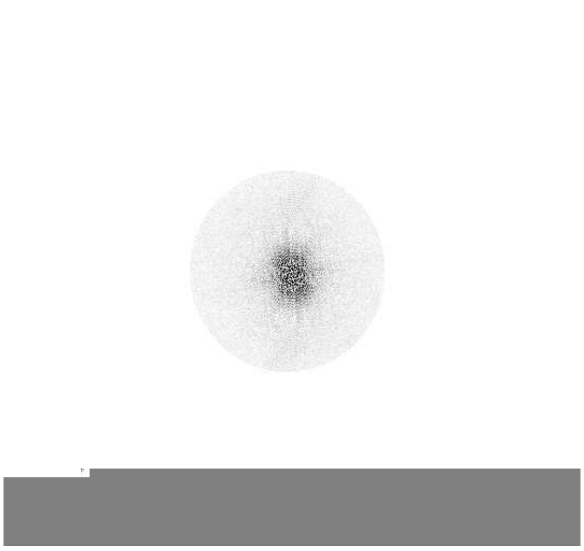
(a) Noisy image.



(b) Denoised.



(c) Fourier transform of noisy image.



(d) Fourier transform after cutoff.

Figure 3.4: Effect of the cutoff filter. Notice the ripples surrounding the borders and the ondulations produced by this filter.

The filtering process of equation (3.2) can be expressed in the Fourier or frequency domain using the transforms of the image $\hat{\mathbf{X}} = \mathcal{F}(\hat{\mathbf{x}})$ and $\mathbf{H} = \mathcal{F}(\mathbf{h})$ as

$$\hat{\mathbf{x}} = \mathcal{F}^{-1}(\hat{X} \odot \mathbf{H}) \tag{3.7}$$

where $\odot$ denotes element-wise product. This formulation has practical and theoretical implicancies. For instance, it can be used to analyze the frequency behavior of a linear filter in a graphical way.

## 3.5 Bounded variation methods

These are methods which impose constrains on the magnitude of the overall fluctuations in the image. Thinking of the denoised image as an $\mathbb{R}^2 \to \mathbb{R}$ function, a solution is found which tries to meet two goals at once: to approximate the clean image as best as possible, and to minimize its *Total Variation* [28]. Roughly speaking, the total variation of an image is a global measure of how much does it change its value from sample to sample. One possible way to define this is by summing the absolute magnitude of its gradient at each position:

The idea is that most of the small fluctuations on the image are due to the noise. By reducing these fluctuations incrementally, a solution can be found in which most of the noise is smoothed out and the bigger fluctuations (borders, etc.) are preserved. The denoising problem is posed as a minimization of a function $\mathbf{F}(\hat{\mathbf{x}}^{m\times n})$,

$$\mathbf{F}(\hat{\mathbf{x}}^{m\times n}) = \sum_{i\in R_{m\times n}} |\mathbf{x}_i - \hat{\mathbf{x}}_i| + \beta \sum_{i\in R_{m\times n}} \sum_{j\in W_{i,T}, j\neq i} \phi(\hat{\mathbf{x}}_i - \hat{\mathbf{x}}_j) \tag{3.8}$$

The first summation in (3.8) is minimized when the denoised image $\hat{\mathbf{x}}^{m\times n}$ is as close as possible to the unknown clean image $\mathbf{x}^{m\times n}$, while the second summation accounts for the total variation measure of the solution. $W_{i,T}$ is a small neighborhood window where the variation of each sample $i$ is measured, and the function $\phi(.)$ models the penalty assigned to high fluctuations. Examples of $\phi(.)$ are $\phi(t) = \sqrt{\alpha + t^2}), \alpha > 0$ or $\phi(t) = |t|^\alpha, 1 < \alpha < 2$.

These algorithms tend to destroy the small details and fine textures present in an image. On the other side, they produce good results when the noise power is high.

## 3.6 Statistical filtering methods

Many image denoising algorithms are derived from the theory of *Statistical Signal Processing* [11]. Under this theory the image (signal) is modeled as a *ramdom process*, i.e., a vector (possibly of infinite length) of random variables. This is implicit in the description of the noisy channels described earlier in this chapter, where the noise is considered to be a sequence of independent and identically distributed (i.i.d.) random variables $\mathbf{n}^{m\times n}$. In fact, any of the previously presented algorithms has a statistical interpretation. This section concentrates on those algorithms which are based on statistical models to produce their output.

The unknown clean image is considered to be a random process itself, and it is expected to exhibit a set of statistical properties (high correlation between samples, repeated patterns) which distinguish it from the properties of the noise process that corrupts it (small or no correlation). Two common assumptions on the properties of clean images are:

- *Markovicity*, which means that each sample, when conditioned on a neighborhood of some (fixed) size, is statistically independent of the rest of the image.
- *Stationarity*, meaning that the statistic properties of the samples of the image are the same for all samples regardless of their position in the image.

Examples of statistical filters are the *Wiener Filter* [35], the Lee Filters [14] (also known as *Local Wiener* filtering), the *Gaussians Scale Mixture* (GSM) filters [25] which will be described later in this chapter, and last but not least the DUDE [34] which forms the basis of the present work.

To fix ideas the classical Lee Filter is described. The Lee Filter estimates each clean sample in a two-step way:

1. Using a fixed-size sliding window $W_{i,T}$, estimate the local mean of the image $\mathbf{z}^{m\times n}$ at each position $i$,

$$\mu_i = \frac{1}{K}\sum_{r=1}^{K} \mathbf{z}(W_{i,T})_r$$

2. Estimate the local variance $\sigma_i^2$ as

$$\sigma_i^2 = \frac{1}{K-1}\mathbf{z}(W_{i,T})^T\mathbf{z}(W_{i,T}) - \mu_i^2$$

   The contexts $\mathbf{z}(W_{i,T})$ are column vectors, $\mathbf{z}(W_{i,T})^T$ are their transposed (row) versions, and $K$ is the size of the contexts.

3. Using $\mu_i$, $\sigma_i^2$ and the noise power $\sigma_n^2$ which is considered constant throughout the whole image, estimate the clean sample as

$$\hat{\mathbf{x}}_i = \frac{1}{\sigma_i^2 + \sigma_n^2}\left(\sigma_i^2\mathbf{z}_i + \sigma_n^2\mu_i\right) \tag{3.9}$$

Equation (3.9) is the minimum expected square error (MSE) solution of $\hat{\mathbf{x}}_i - \mathbf{x}_i$ given $\sigma_i^2$, $\sigma_n^2$ and $\mu_i$. Both the local mean and variance are derived from the Markov assumption since they are computed only from the local context. The second assumption is not used in this filter.

The Lee filter gave rise to many other algorithms which combine optimization, statistics and structural priors to obtain optimal estimations of the clean image, includinig those which comprise the current state of the art as is the case of the GSM-based algorithms [25].

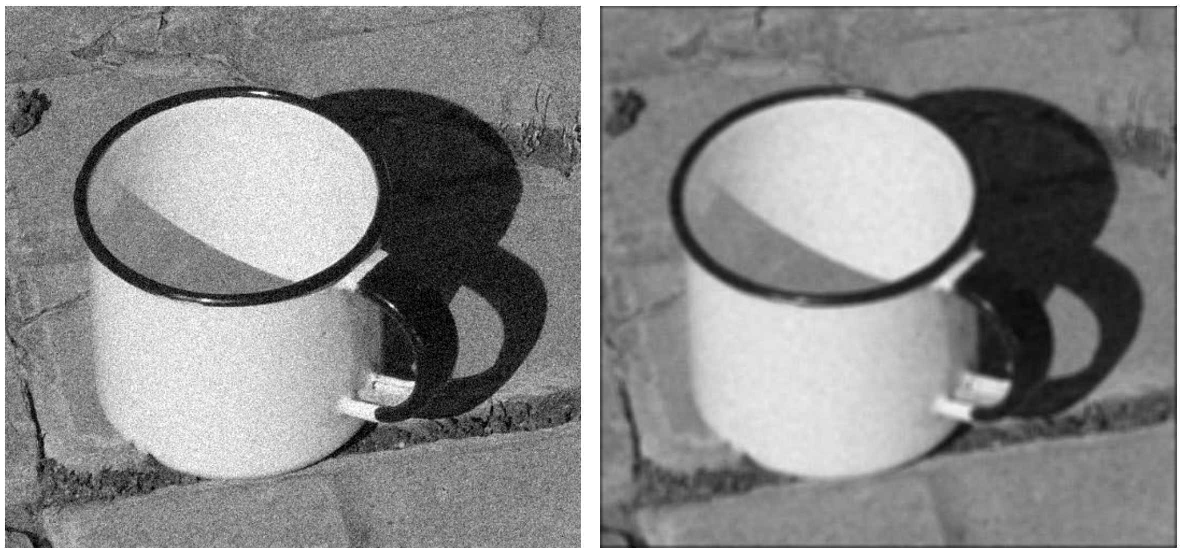

(a) Noisy image. (b) Denoised by average filtering, window size $5 \times 5$.

Figure 3.5: Average Filter of an image corrupted by gaussian noise of $\sigma = 20$.

## 3.7 Filters for additive noise

### 3.7.1 Window average

This is the simplest way to reduce the amount of additive noise in an image. Given a window template $T$ of size $k = |T|$, each pixel is substituted by the average of the values within the window centered on it.

$$\hat{\mathbf{x}}_i = \frac{1}{k}\sum_{r=1}^{k} \mathbf{z}_{j_r} \tag{3.10}$$

This is a special case of (3.2) where $h_r = \frac{1}{k}, \forall r$ . As the noise is considered additive, $\mathbf{z}_i = \mathbf{x}_i + \mathbf{n}_i$ and

$$\hat{\mathbf{x}}_i = \frac{1}{k}\sum_{r=1}^{k} \mathbf{z}_{j_r} = \frac{1}{k}\sum_{r=1}^{k} \mathbf{x}_{j_r} + \frac{1}{k}\sum_{r=1}^{k} \mathbf{n}_{j_r} \tag{3.11}$$

Here the first summation will be close to the clean value if the clean samples in the window are also similar, and the second summation will converge to the expectation of the noise which is 0 as $k$ increases. If the pixels in the window are not similar (which happens in borders and high contrast areas), the details of the image are blurred. This effect increases with the size of the window, which implies a tradeoff between noise remotion and detail preservation in terms of $k$. Figure 3.5 shows the result of this filter for a square window of $5 \times 5$ pixels.

This example shows the motivation behind each of the algorithms in subsections 3.7.2 through 3.7.6 : how to remove the noise without destroying the details of the image?

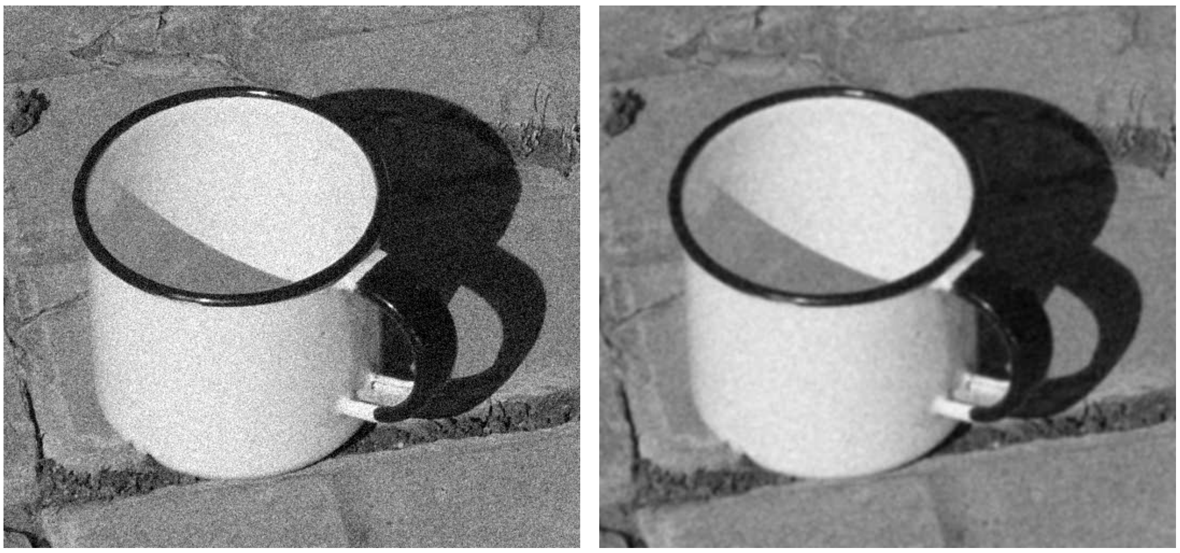

(a) Noisy image. (b) Denoised by a Gaussian Isotropic Filter, window size $5 \times 5$.

Figure 3.6: Effect of the Gaussian Filter ($w = 1.4$) on an image corrupted by gaussian noise with $\sigma = 20$.

## 3.7.2 Isotropic gaussian filtering

A plain average of the window samples is generally not a good solution. There are two main reasons for this: first, the assumption that the neighboring samples are similar to the center sample becomes weaker as the distance from the center increases. Second, the frequency response of the average filter is not as in a sharp cutoff filter, but decays slowly and is significant all over the frequency spectrum including those parts where the noise is high and the image power is low, leading to undesired high frequency effects in the image (blocking). The idea is to solve the first problem by giving more weight to the samples which are nearer to the center pixel, and less weight to the ones which are farther. In principle, every pixel in the image is taken into account, but practical implementations usually approximate them as linear fixed window filters.

An *isotropic* filter assigns the weight of each sample of the image based only on its euclidean distance to the center pixel $\|i - j\|_2$. If a linear window filter is used, the window kernel terms $h_r$ are obtained using the corresponding index offsets $d$ in place of $i - j$. One common choice to assign the weights is the 2D Gaussian kernel $G_w$:

$$G_w(\cdot) = \frac{1}{4\pi w} e^{-\frac{\|\cdot\|^2}{4w^2}} \tag{3.12}$$

in which case this is called a *Gaussian filter*. The paramenter $w$ controls the radius of the Gaussian kernel and defines the tradeoff between noise removal and detail preservation for this case. Figure 3.6 shows a sample image denoised by Gaussian filtering.

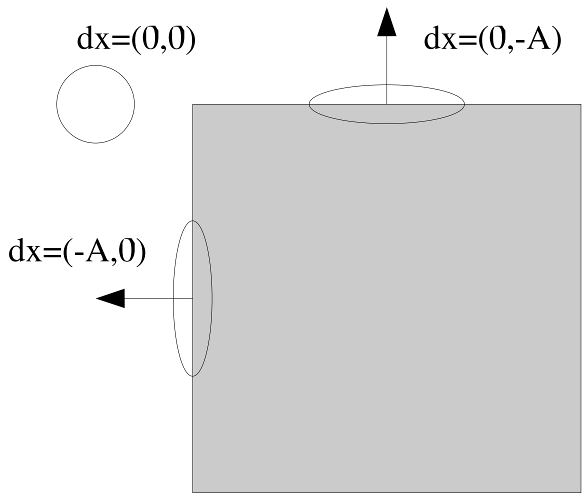


Figure 3.7: Scheme of the basic edge-preserving anisotropic filtering concept.

## 3.7.3 Anisotropic filtering

The Gaussian filter is able to solve the first problem of the window average filter: the high frequency artifacts. However, it does not solve the problem of detail and border preservation. The *anisotropic* filters, as the term implies, assign the weights considering the distance but also a preferred *direction* of filtering.

The basic idea was described in [23], where the direction of filtering is determined by the output of a local edge detector. By modifying the shape of the kernel according to the local gradient, the kernel assigns more weight to the pixels "along" the gradient and less weight to the pixels "across" the gradient so that the filter does not "cross the borders". This behavior is depicted in Figure 3.7. Let $\mathbf{dx}_1$ and $\mathbf{dx}_2$ the vertical and horizontal derivatives of $\mathbf{x}^{m\times n}$. Let $\nabla x(i) = (\mathbf{dx}_1, \mathbf{dx}_2)$ denote the gradient of $\mathbf{x}^{m\times n}$ at index $i$. Using the more general definition of the Gaussian kernel

$$G_w(.) = \frac{1}{2\pi|\Sigma|^{1/2}} e^{-\frac{(i-j)^T\Sigma^{-1}(i-j)}{2}} \tag{3.13}$$

where the eigenvalues and eigenvectors of the matrix $\Sigma$ control the shape and orientation of the kernel. The matrix $\Sigma$ is constructed so that the two eigenvectors $\theta_1$ and $\theta_2$ are in the direction of the gradient (the normal direction ) and the tangent,

$$\begin{aligned} \theta_1 &= \nabla x_i/|\nabla x_i| \\ \theta_2 &= \nabla x_i^{\perp}/|\nabla x_i| \end{aligned}$$

and the respective eigenvalues $\lambda_1$ and $\lambda_2$ are proportional to the stretching along each of these directions,

$$\lambda_1 \propto \frac{1}{|\nabla x_i|}$$
$$\lambda_2 \propto |\nabla x_i|$$

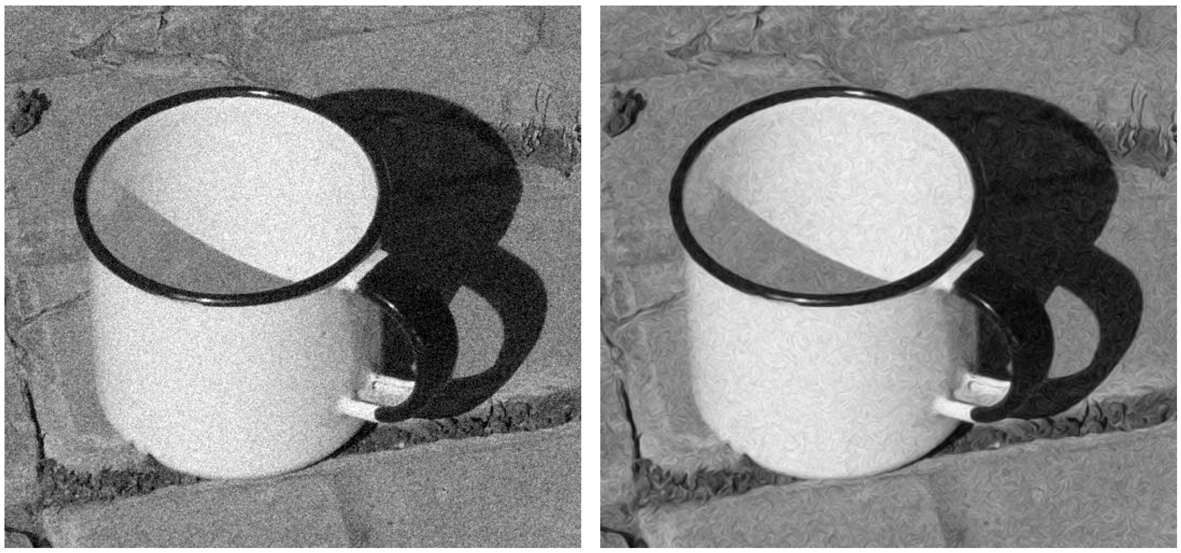

(a) Noisy image. (b) Denoised by PDE anisotropic filtering.

Figure 3.8: Effect of the GREYCstoration Anisotropic Filter on an image corrupted by gaussian noise with $\sigma = 20$.

.

The resulting matrix has the following form:

$$\Sigma = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \times \begin{bmatrix} \frac{dx}{|\nabla x_i|} & \frac{-dy}{|\nabla x_i|} \\ \frac{dy}{|\nabla x_i|} & \frac{dx}{|\nabla x_i|} \end{bmatrix}$$

The value $\lambda_2$ can be chosen so that $|\Sigma|$ is constant (which means that, roughly speaking, the "area" of the kernel is always the same), or $|\Sigma| \propto |\nabla x_i|$. One common choice is $\lambda_1 = e^{-|\nabla x_i|^2/\sigma}$ where $\sigma$ is a threshold above which the kernel starts to stretch and avoids the effect of the noise itself in the value of $|\nabla x_i|$. In this case, the amount of denoising is controlled by $|\Sigma|$.

Another way of performing anisotropic filtering is by using PDEs (Partial Differential Equations) [29]. When used for denoising, PDEs are able to define anisotropic behaviors which depend on features more complex than local borders such as local curvature [31]. The image in Figure 3.8 was obtained using a curvature-driven anisotropic PDE filter, made publicly available by the author in the form of a GIMP (GNU Image Processor) plugin.[1]

### 3.7.4 Non-Local Means

All of the previous filters use *local* information to compute the denoised pixels. The *Non-Local Means* is a recent method to remove additive noise and is described in full detail in [1] (which also serves as a good review of additive noise removal algorithms including many not listed here).

[1]http://www.gimp.org/
http://www.haypocalc.com/wiki/Plugin_Gimp_GREYCstoration as of August 2005

Here, in contrast to the previous filters, each denoised pixel $\hat{\mathbf{x}}_i$ (the target) is obtained as a weighted average of *all* the other pixels of the image, where the weight of each pixel is determined by a measure of similarity between its neighborhood and the neighborhood of the pixel to be denoised:

$$\hat{\mathbf{x}}_i = \frac{\sum_{j \in R_{m\times n}, j\neq i} w_{ij}\mathbf{z}_j}{\sum_{j \in R_{m\times n}, j\neq i} w_{ij}}. \tag{3.14}$$

Let the operator $*$ denote inner vector product. The weights are defined as

$$w_{i,j} = f\left(G_a(W_{i,T}) * |\mathbf{z}(W_{i,T}) - \mathbf{z}(W_{j,T})|\right) \tag{3.15}$$

where $f$ is a monotonically decreasing function, usually $e^{-\frac{x^2}{2w}}$ for some $w > 0$ and $G_a$ is a 2D Gaussian kernel of parameter $a$ which weights the difference of the samples at each location according to their distance to the center of the window. If $N$ is the number of pixels, this algorithm requires $O(N^2)$ operations to produce a result, which makes it impractical for medium sized images as originally proposed. However, it gives very good results and serves as a reference for other denoising algorithms. Figure 3.9 shows some examples taken directly from [1].

### 3.7.5 Wavelet thresholding

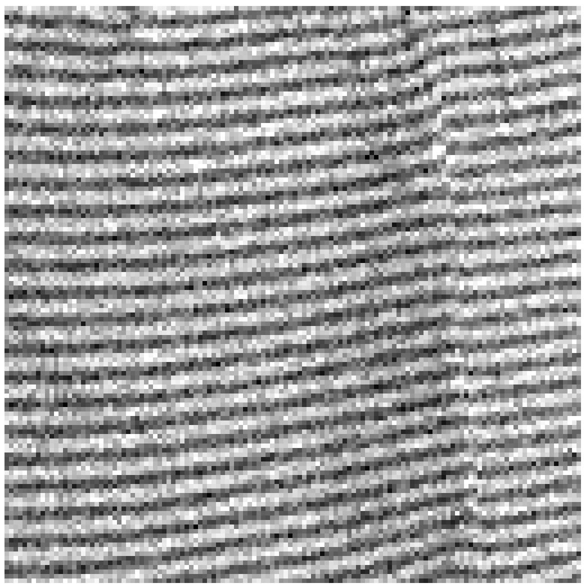
The name *Wavelets* refers to a general family of transforms whose characteristic is to combine spatial and frequential information in the transformed data [16][5]. As with the frequency (Fourier) domain filters, the idea is to concentrate the information of the "true" clean image in some coefficients, and discard or atenuate the coefficients which are more affected by the noise addition process. The *Wavelet thresholding* method [6] does this by simply discarding all those coefficients which are below a certain threshold and reconstructing the image with the remaining coefficients. Some enhancements to the basic idea have been proposed [3]. In particular, the *Wavelet-Curvelet thresholding* [30] gives results comparable to the state of the art for this type of noise, at least for the Lena image. Figure 3.10 shows the results published in [30].

### 3.7.6 Mixture of gaussians

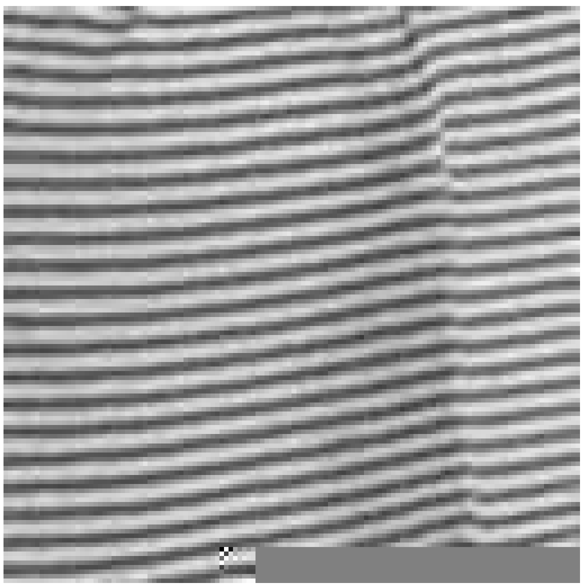
This is another wavelet-based approach, although very different from the one previously described. Three novel elements appear in this algorithm:

- An *overcomplete* decomposition of the image into what is called a *steerable pyramid.* It is overcomplete because the resulting representation has more samples than the original image (this does not happen with ordinary transforms such as Fourier decompositions or orthogonal wavelets).

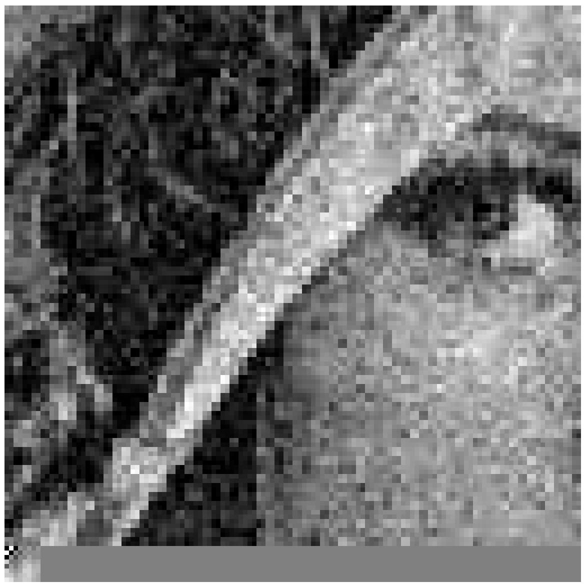
- A probability model of the coefficients of the pyramid based on a *Gaussian Scale Mixture* (GSM) probability distribution. This model assumes Markovicity in terms of 3D neighborhoods in the pyramid and uses the GSM to model each vector of neighborhood coefficients. The GSM is a generalization of the multivariate Gaussian distribution. A vector $\mathbf{v}$ is distributed according to a GSM if $\mathbf{v} \overset{d}{=} \sqrt{z}\,\mathbf{u}$ where $\overset{d}{=}$ means equality in distribution, $\mathbf{u} \sim N(0,\Sigma)$ and $z$ is a scalar multiplier obeying some other arbitrary distribution.



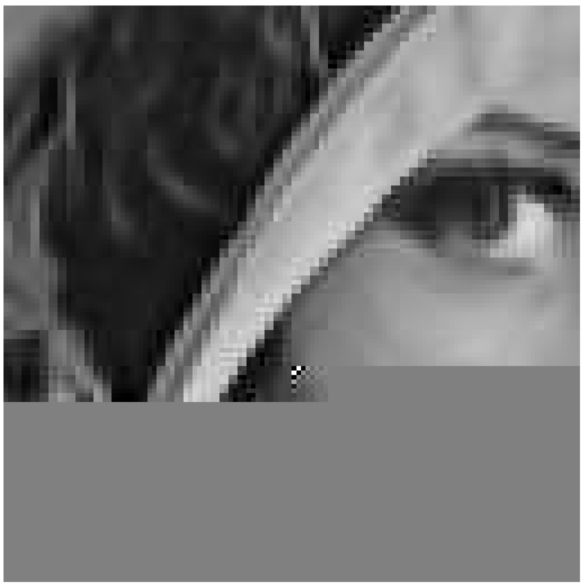
(a) Noisy image.



(b) Denoised by NLM.

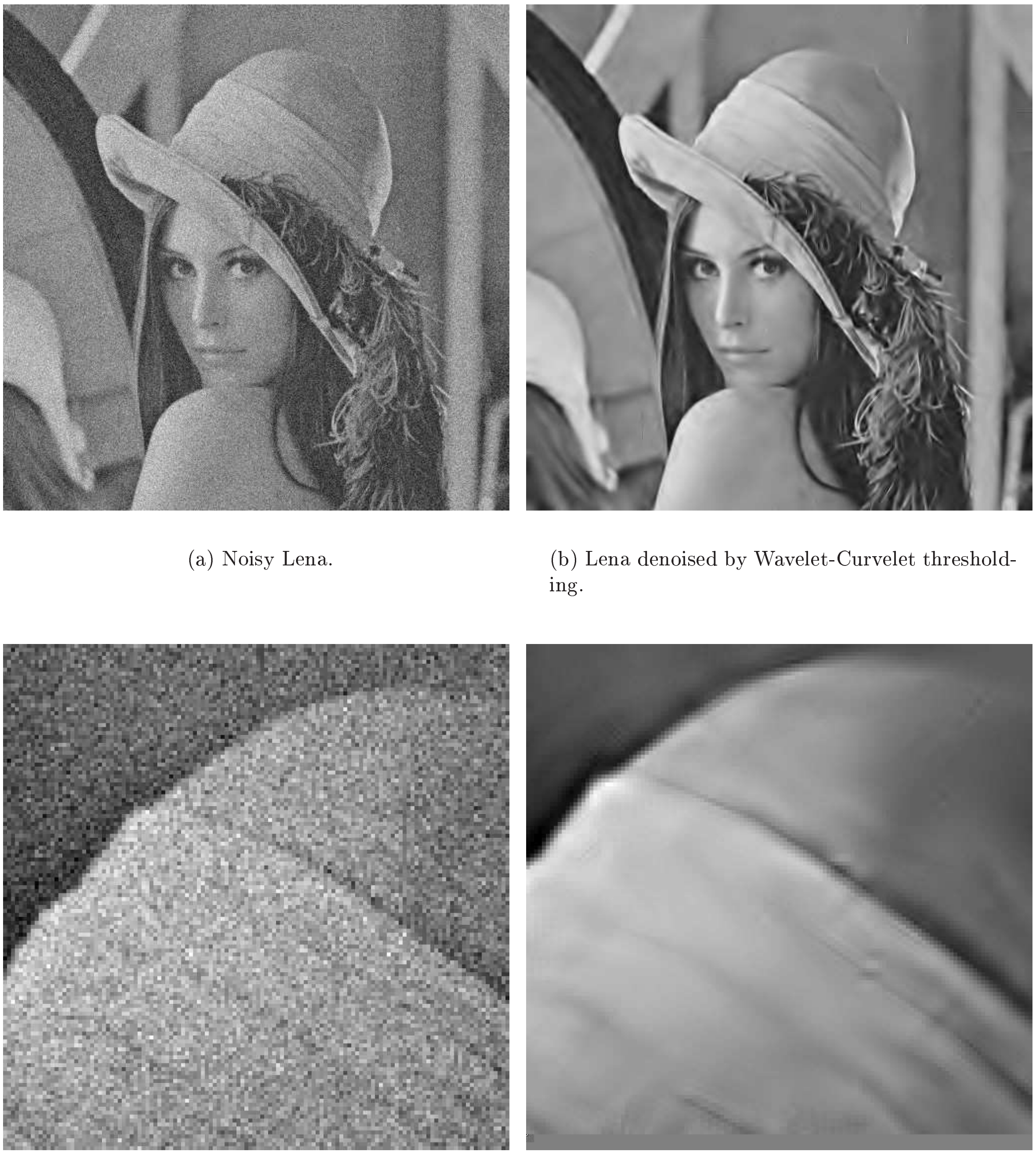


(c) Noisy image.



(d) Denoised by NLM.

Figure 3.9: Effect of the NLM Filter on two images corrupted by gaussian noise with $\sigma = 20$. The results are obtained using the whole images (barb and lena respectively), although only a small representative patch is shown.



(a) Noisy Lena.

(b) Lena denoised by Wavelet-Curvelet thresholding.

(c) Detail of (a).

(d) Detail of (b).

Figure 3.10: Effect of the Wavelet-CurveletThresholding Filter on “Lena” corrupted by gaussian noise with $\sigma = 20$.

Having the noisy image decomposed as a steerable pyramid, and a model for the noise, the algorithm proceeds much like the Lee Filter, estimating the parameters of the GSM for each context and then computing an expected least squares error estimate for the denoised output sample. The details of the algorithm are beyond the intentions of this intruduction. Please refer to [25] for these and for other references regarding GSMs and GSM-based denoising.

## 3.8 Filters for non-additive noise

Non-additive noise channels have two properties which are exploited by all of the algorithms described in subsections 3.8.1 — 3.8.4:

- An important fraction of the pixels in the noisy image are left untouched, i.e., have the same value as the corresponding clean pixels.
- The noisy pixels have no correlation with the corresponding clean (unknown) pixels.

These two facts are used to *detect* the noisy pixels and sepparate them from the clean pixels, and to estimete these noisy pixels with a few clean neighboring pixels.

### 3.8.1 Median filter

The idea of this filter is very similar to that of the average filter. As with the average filter, this is a fixed sliding-window algorithm which depends on a window template $T$ yielding different windows $\mathbf{z}(W_{i,T})$ for each index $i$. Because the non-additive noise samples take on arbitrary values, a window would contain many *outliers* (samples very different in value with the majority of the samples in the window), and the average of the samples would not be a good estimate. Instead, the *median* estimator ($\mathrm{m}ed(.)$) of the window samples is used, as it is more robust to the presence of outliers. The median estimator of a vector of samples $z(W_{i,R})$ is computed as follows:

- Order the samples of the vector $z(W_{i,T})$ in decreasing (or increasing) order. Call this vector $m$.
- Let $k = |T|$ be the size of the vector. If $k$ is even, $\hat{\mathbf{x}}_i = \frac{1}{2}\left(m_{k/2} + m_{k/2+1}\right)$; otherwise $\hat{\mathbf{x}}_i = m_{(k+1)/2}$.

This filter does not use the first property explicitly, which means that all the pixels of the resulting image are the result of their window median. As with the average filter, this results in a blurring effect (although non-linear), with the same tradeoffs implied . The following algorithms try to use this information to improve the results.

### 3.8.2 Selective median (basic)

The *selective median* approach can be considered a general enhancement to the previous filter which tries to keep those pixels which were not modified by the channel. The problem of finding out which pixels are clean and which are noisy can be attacked in various ways. For example, if the noise is impulsive such as in the Salt & Pepper case, the values of the noisy pixels are known *a priori*, and a trivial scheme can be implemented in which only those pixels in the noisy image which have the maximum value (white) or the minimum value (black) are substituted by the median of the window.

$$\hat{\mathbf{x}}_i = \begin{cases} med(W_{i,T}) & , \quad \mathbf{z}_i = 0 \; or \; \mathbf{z}_i = M-1 \\ \mathbf{z}_i & , \quad otherwise \end{cases} \tag{3.16}$$

Another slightly more robust approach is to consider as noisy all those pixels above or below a certain threshold. In this case

$$\hat{\mathbf{x}}_i = \begin{cases} med(W_{i,T}) & , \quad \mathbf{z}_i \leq \tau \; or \; \mathbf{z}_i \geq M-1-\tau \\ \mathbf{z}_i & , \quad otherwise \end{cases} \tag{3.17}$$

## 3.8.3 Adaptive Median

The basic selective median filter uses a fixed window to denoise each noisy pixel. The noisy pixels are previously detected using any of the previously described methods. The adaptive median [12] chooses an optimal window size depending on how many noisy pixels there are in the neighborhood, starting with a square $3 \times 3$ window and increasing its size gradually until a fixed maximum. For each noisy pixel $\mathbf{z}_i$ The algorithm can be summarized as follows:

```
· Initialize $w = 3$
· Compute $a = \min(z(W_{i,w\times w}))$, $m = median(z(W_{i,w\times w}))$ and $b = \max(z(W_{i,w\times w}))$
· If $a < m < b$ go to Step 5, otherwise set $w = w + 2$.
· If $w < w_{max}$ go to step 2, otherwise set $\hat{\mathbf{x}}_i = m$.
· If $a < \mathbf{z}_i < b$ set $\hat{\mathbf{x}}_i = \mathbf{z}_i$, otherwise set $\hat{\mathbf{x}}_i = m$.
```

Figure 3.11: Adaptive Median Algorithm.

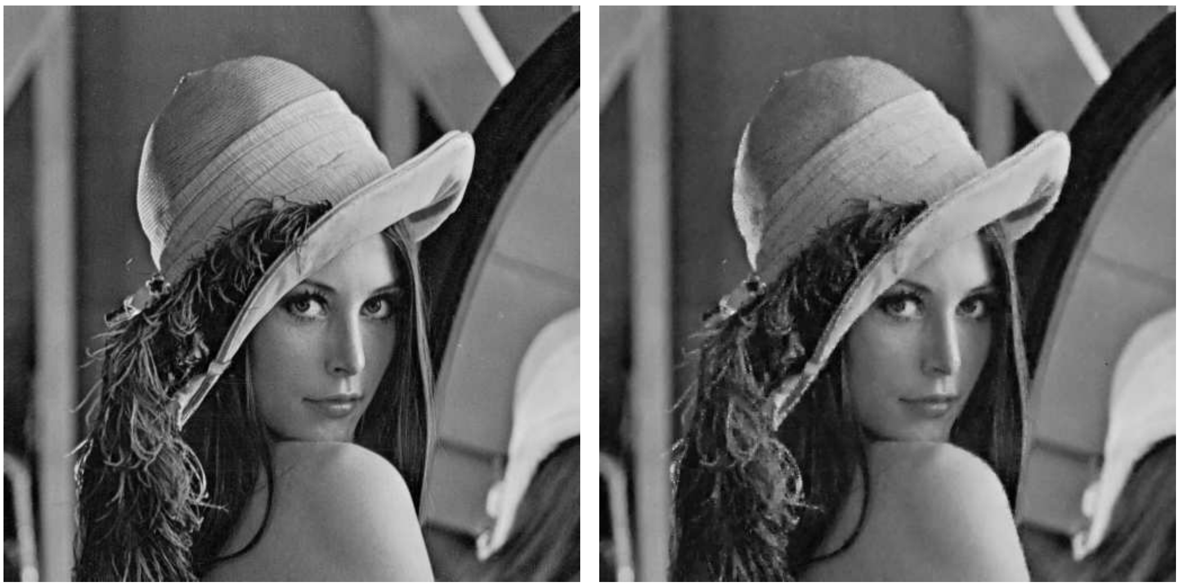
This type of filter is usually suitable for images corrupted with Salt & Pepper noise with high probability of error $\lambda$.

## 3.8.4 Adaptive Median and Total Variation Combined

The idea of this scheme, as proposed in [2], is to combine the Adaptive Median scheme with the Total Variation approach described earlier in this chapter. The pixels of the image are divided into two groups using the selection criterion of Algorithm 3.11: the noisy $\mathcal{N}$ and the clean $\mathcal{N}^c$ (both groups are defined in terms of the indexes of the image). Then, (3.8) is used with a slight modification:

$$\mathbf{F}_{\mathcal{N}}(\hat{\mathbf{x}}^{m\times n}) = \sum_{i\in\mathcal{N}} |\mathbf{x}_i - \hat{\mathbf{x}}_i| + \beta_1 \sum_{i\in\mathcal{N}} \sum_{j\in W_{i,T} \bigcap \mathcal{N}, j\neq i} \phi(\hat{\mathbf{x}}_i - \hat{\mathbf{x}}_j) + \beta_1 \sum_{i\in\mathcal{N}^c} \sum_{j\in W_{i,T} \bigcap \mathcal{N}^c, j\neq i} \phi(\hat{\mathbf{x}}_i - \mathbf{z}_j) \tag{3.18}$$

Here the last two summations correspond to the total variation. The first of these is expressed in terms of variation between noisy samples, while the second measures the variation of the noisy samples with respect to the clean ones. The overall expression is also constrained only to those indexes $i$ that correspond to noisy pixels. Figure 3.12 shows two example images obtained by this method, taken from [2].



(a) Noisy image. (b) Denoised by MDN-DP.

Figure 3.12: Effect of the MND-DP Filter on an image corrupted by Salt & Pepper noise with $\lambda = 70\%$. Image taken from [2].

# 4 The Discrete Universal DEnoiser

## 4.1 Description of the algorithm

The Discrete Universal DEnoiser (DUDE) algorithm [34] operates over the noisy output sequence of a known discrete memoryless channel, estimating the noiseless input sequence to that channel without any assumption on the statistical properties of this noiseless input sequence. This algorithm has been shown to achieve asymptotically the optimal finite sliding window denoiser performance for any input sequence as the length of the sequence goes to infinity.

Here is a brief outline of the algorithm, full details of which can be found in [34].

For clarity the DUDE is described for the case where the unknown image $\mathbf{x}^{m\times n}$ has an associated "clean" probability distribution (stochastic setting), although the results also apply for the case where $\mathbf{x}^{m\times n}$ is an individual image not assumed to have been emmited by a stochastic source (semi-stochastic setting).

The DUDE operates in two passes: An analysis pass and a denoising pass. Both passes are parameterized by the same neighborhood template $T$. In the theoretical analysis of [34], the size of the template grows with the length of the data, and has to obey certain growth rate restrictions to guarantee the asymptotic convergence of the algorithm to the optimal denoisability. This is discussed in [34, Sec VII-A] for one-dimensional data, and in [21, Sec. 3] for two-dimensional (2D) images.[1] However, the determination of the exact size (and shape) of $T$ that yields the optimum denoiser performance for a *given* image is a difficult open problem. Possible approaches to the problem are discussed in [34, Sec. VII], together with a *compressibility* heuristic which is also employed in [21, Sec. VII-B], and in this work in Section 7. More recently, an approach for optimizing context size based on an estimate of the residual noise after application of the DUDE was presented in [22].

**The first pass** uses a sliding neighborhood window $W_{i,T}$ to determine the context $C_i = \mathbf{z}(W_{i,T}), C_i \in \mathcal{A}^K$ of each pixel $\mathbf{z}_i$. For each different context $C$ appearing in in the image, a vector of statistics $m_C$ is built where $m_C[i]$ counts the occurences of all the values of $\mathbf{z}_i$ whose context $C_i$ is equal to $C$. Note that $|m_C| = |\mathcal{A}|$.

**Input probability estimation**

[1]For the case of 2D images over an alphabet of size $M$, and using $L_2$ (Euclidean distance) balls of radius $r$ as the template shape, the asymptotic optimality as the size of the image $m \times n$ grows to infinity is guaranteed if $r$ has the form $r(m,n) = g(\min\{m,n\})$ where $g(t)M^{g(t)} = o(t^{1/4})$. For instance, a choice of $r = g(t) = c \log_M t$, with $c < 1/4$ satisfies the requirement.

After the first pass is done, each statistics vector $m_C$ is normalized to yield an estimated context-conditional output distribution $P_{Z|C}$, which is a row vector of size $|\mathcal{A}|$ where $P_{Z|C}[i] = P_{Z|C}(Z = i)$,

$$P_{Z|C}(Z = i) = \frac{m_C[i]}{\sum_{j \in \mathcal{A}} m_C[j]}, \forall i \in \mathcal{A} \tag{4.1}$$

By knowing the channel through its transition matrix $\Pi$ and its memoryless nature, the DUDE is then able to estimate the correspoding clean sequence context-dependent distribution $P_{X|C}$ for each context $C$ by solving the following linear system

$$P_{X|C}\Pi = P_{Z|C} \tag{4.2}$$

After the context-conditional input probability is estimated, the next step is to condition it also on the noisy sample, $\alpha$. Using $\odot$ as the vector element-wise product operator, the resulting distribution can be shown to be

$$P_{X|C,\alpha} = \frac{1}{P_{Z|C}(\alpha)} P_{X|C} \odot \pi_\alpha. \tag{4.3}$$

With these elements, a denoiser function is then defined which minimizes the expected loss for each possible combination of the context $C$ and the noisy symbol $\alpha$. The term $P_{Z|C}(\alpha)$ is dropped from (4.3) since it doesn't depend on the minimizing argument, to obtain

$$g(\alpha, C) = \arg\min_{a \in \mathcal{A}} (P_{X|C}[\lambda_a \odot \pi_\alpha]) \tag{4.4}$$

(note that $\lambda_a \odot \pi_\alpha$ is a column vector, and $P_{X|C}$ is a row vector, thus the preceding expression is the inner product of the two).

If the channel is invertible, the above expression becomes

$$g(\alpha, C) = \arg\min_{a \in \mathcal{A}} (P_{Z|C}\Pi^{-1}[\lambda_a \odot \pi_\alpha]) \tag{4.5}$$

**The second pass** of the DUDE applies the denoiser function (4.4) based on the statistics gathered in the first pass for each observed context $C$.

The algorithm is summarized in Figure 4.1.

The DUDE has been applied to binary (1 bit per pixel) images [21] outperforming other existing denoising schemes for this type of data. Figure 4.2 shows a sample result performed on a halftone image transmitted over a simmulated Binary Symmetric Channel. This channel flips each sample bit value with probability $p$, and leaves it untouched with probability $1 - p$. In this case, $p = 2\%$.

```
· Initialization: For each possible context $C$ that can arise from a window defined by the
  template $T$, define a vector of counts of size $|\mathcal{A}|$ and initialize its elements to 0.
· Pass 1: for each pixel $\mathbf{z}_i$
    · Obtain the current context $C = \mathbf{z}(W_{i,T})$ using the neighborhood template $T$ and index
      $i$.
    · Increment $m_C[\mathbf{z}_i]$.
    · Normalize $m_C$ for each possible context $C$ to yield $P_{Z|C}$ using (4.1).
· Pass 2: for each pixel $\mathbf{z}_i$
    · Obtain the current context $C = \mathbf{x}(W_{i,T})$ using the neighborhood template $T$ and index
      $i$.
    · Compute $P_{X|C}$ using the channel transition matrix $\Pi$ and (4.2).
    · Compute $P_{X|,C,\mathbf{z}_i}$ using the loss matrix $\Lambda$ and (4.3) with $\alpha = \mathbf{z}_i$.
    · Compute the denoised pixel using (4.4).
```

Figure 4.1: Baseline DUDE algorithm.

## 4.2 Issues of the DUDE with continuous tone images

The asymptotic optimallity of the DUDE applies to images whose symbols range over any finite alphabet. However, this asymptotic behavior is governed by a decay term which increases rapidly with the size of the alphabet and the size of the context window.

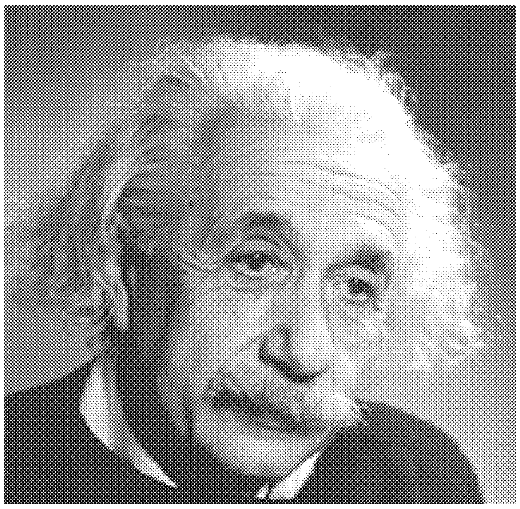
Sequences such as digital images or audio tracks are finite and have alphabets whose size range from 256 (8 bits) to 65536, or even 16 million symbols for audio signals. If such sequences were normally long enough for the DUDE to perform well even with the slow convergence implied by the size of such alphabets, then there would be no problem in applying it as originally proposed. Unfortunately, this is not the case and the optimal performance will not be achieved.

These kind of sequences are normaly drawn from continuous processes that are later discretized. As a result, continuous-tone images have structural properties that can be incorporated as prior knowledge in the denoising process to avoid the mentioned problems.

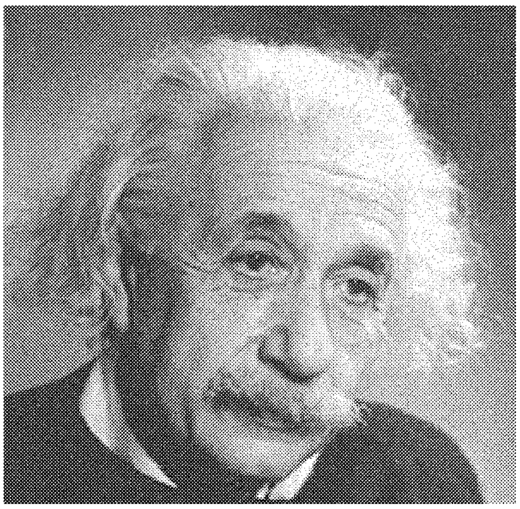
## 4.3 Goal of this work

The goal of this work is to augment the DUDE framework, by including the prior knowledge derived from the structure of continuous-tone images, so that it can be applied to this kind of data with success.

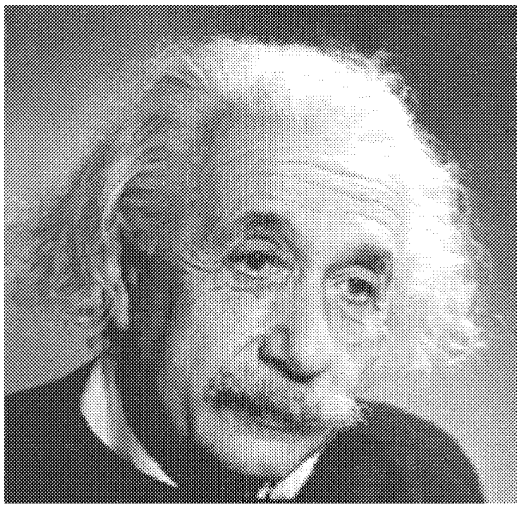
This primary goal is to be met while keeping the framework efficient in terms of computational cost, resulting in a practical implementation.



(a) Clean image.



(b) Noisy image.



(c) Denoised by the binary DUDE.

Figure 4.2: Images obtained from [21]. Here the window template is a line of 7 samples to each side of the center sample (a $1 \times 15$ template)

# 5 Tools

## 5.1 Context modeling

The main goal of this work is to exploit the a priori information about the structure of continuous-tone images (piecewise continuity, repeated texture patterns, etc.) in order to reduce the convergence problems that arise when applying the original DUDE algorithm to sequences with such large alphabets.

To denoise an image, the DUDE relies on the conditional distributions estimated in the first pass for each context. Determining conditional distributions of samples given their contexts is also a key component in lossless data compression where the number of conditioning contexts plays a fundamental role in the convergence of the code length to the entropy. This code length includes either implicitly or explicitly a *model cost* [26] which is proportional to the number of free statistical parameters in the model.

The model cost reflects the price paid for learing the statistics of the data: if there are many parameters to estimate, more data samples will be required to accumulate significant statistics for each parameter (hence the problem is sometimes described as one of "sparse statistics").

The model cost is particularly affected by the size of the alphabet, as it affects both the potential number of different contexts and the number of parameters per context.

The other component of the code length, a *model fitness component*, is determined by the degree to which the elements of the model (the parameters) capture the statistical properties of the data (i.e., how does it "fit" the data). From the theory and practice of universal lossless compression arises the fundamental trade-off between the two components: a richer model can fit the data better, yielding a shorter model fitness component at the expense of a greater model cost component.

In denoising, and particularly in the DUDE, there exists a similar trade-off. This trade-off is described in [34] in terms of the context size. Given the size of an image, a greater context size implies less occurences of each context in the image, thus reducing the average number of available samples to describe each conditional distribution. This results in a "denoising model cost", where the price paid in this case is a poorer denoising performance.

The number of contexts, and the number of parameters per context increase as the alphabet size grows. For instance, in the DUDE, the number of possible contexts grows as $\mathcal{A}^K$ and the size of each context-conditional count vector gathered grows linearly with $\mathcal{A}$. This results in a total of $O(\mathcal{A}^{K+1})$ parameters to be estimated in the model produced by the first pass of the DUDE.

To address this problem, prior information on the structure of continuous-tone images is used to let contexts *share their information* , allowing the statistical information of many contexts to contribute in the estimation of the conditional distribution at each image location.

As the problem of modeling in continuous-tone images has been treated extensively in the field of lossless image compression, it is natural to borrow techiques from this field to address the same problem in denoising. In lossless image compression, two techniques are often used:

**context clustering** partitioning the space of contexts $\mathcal{A}^K$ into a much smaller set of *conditioning classes* $\Gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_N\}$, where the contexts in each class are related by a certain similarity criterion. A *context classifier* $G : \mathcal{A}^K \to \Gamma$ is defined which maps each *raw* context into one of the context classes:

$$\gamma = G(C), C \in \mathcal{A}^K, \gamma \in \Gamma$$

**prediction** exploits the assumption that groups of conditional distributions depend on the conditioning context only through a *context-dependent offset*, given by the predicted value.

These techniques are used, for example, in state-of-the-art compression schemes such as [33] and [36].

The problem of model cost in the DUDE is addressed by augmenting its baseline algorithm to include two additional components: a prediction component and a context clustering component.

Once the context classifier is defined, the sets of counts of all the contexts that are assigned to the same class are added together to build a single class-conditional distribution per class $\gamma$ which we will denote as $P_{Z|\gamma}$. The pixels of the image whose contexts belong to the same class $\gamma$ will be said to have the same *conditioning class* or *state* $\gamma$.

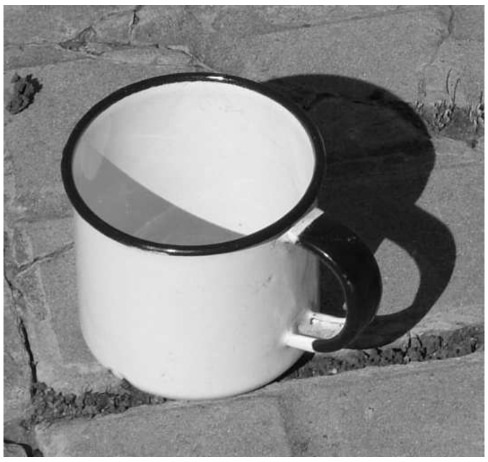
## 5.1.1 Similarity criteria and the structure of images

In order to define the context classes, a context similarity criterion is defined using following common *a priori* assumptions on the structure of continuous-tone images:

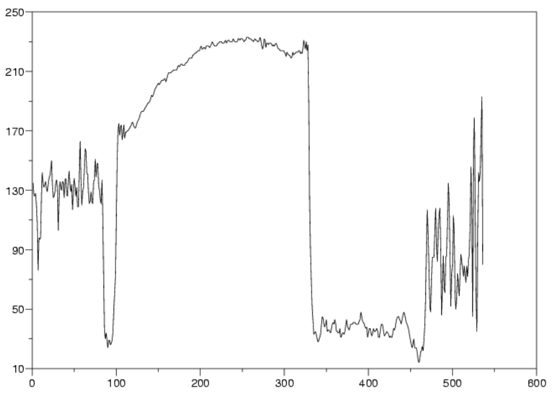
**Distance between symbols** As they represent physical magnitudes (light intensity), the symbols are ordered by value and a distance can be defined between them. For example, Figure 5.1 shows an image and the intensity graph of one of its rows as a $\mathbb{N} \to \mathbb{N}$ function.

**Distance in context space** As the contexts are made of samples, and each sample is an integer magnitude, a distance can be defined in the context space (for example Euclidean distance).

**Local intensity or DC offset** Contexts are localized in the image around a reference pixel. As the image can exhibit similar structures under different local illumination levels, contexts that arise from these structures can also be related if the local illumination level is removed from them. This is known as *DC cancellation*. Figure 5.2 gives an example of this concept.



(a) Continuous-tone image.



(b) Pixel value (intensity) graph for row 130.

Figure 5.1: "Coffee Cup", a continuous tone image of 512 × 512 pixels. The graph to the right corresponds to the pixel values in row 130 starting from the upper row (at about 1/3 of its height).

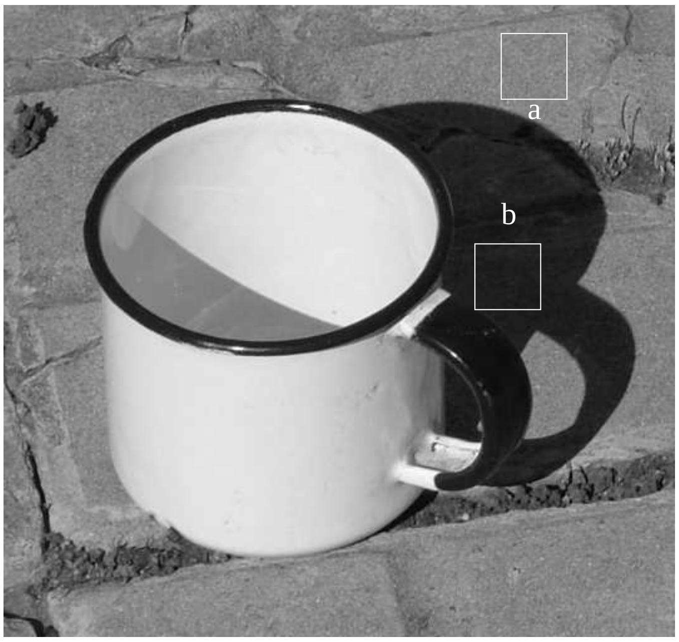


Figure 5.2: DC offset: (a) and (b) are two similar context appearing at different illumination levels. (a) appears in the shadow, while (b) is hit by direct light.

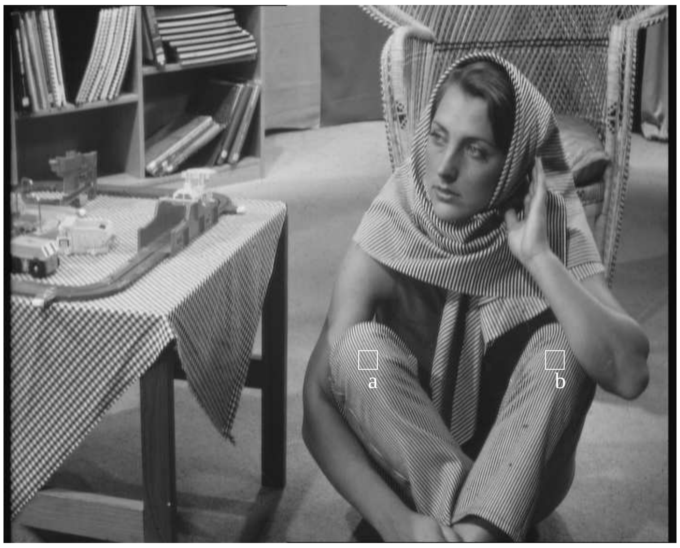


Figure 5.3: Spatial position : (a) and (b) are two similar context appearing at different orientations.

**Spatial position** Contexts can be considered to be "rotation-independent", as many natural and artificial images show similar patterns repeated at different orientations. By exploiting this assumption, contexts can be rotated or scaled before computing the distance between them. As with DC offset, this can result in a smaller number of classes needed to describe the contexts of the image. Figure 5.3 shows two sample contexts which differ only in their orientation.

The way in which these (and possibly other) principles are combined to form a "useful" context model depends on the particular context clustering scheme. If the contexts are to be used to build context conditional probabilities, "useful" means to merge contexts which share "similar" statistics.

## 5.2 Prediction

A predictor is defined as a mapping from the set of possible contexts $\mathcal{A}^K$, to the image alphabet $\mathcal{A}$, $y = p(C), p : \mathcal{A}^K \to \hat{z}$. Predictors can have a fixed structure (for example, one possible predictor is the mean of the context samples) or can vary as a whole or in part depending on the actual context around the pixel to be predicted. The latter are called *context-dependent predictors.*

Images have discontinuities (borders, edges), but in many cases the majority of the pixels belong to smooth areas whose intensity vary with, for example, different illumination angles. By exploiting this fact, it is possible to predict a pixel using a function of a few neighbors.

This makes it possible to gather the empirical statistics of the image in terms of prediction errors (residuals) instead of the original sample values. Furthermore, if the predictor is accurate, then the prediction errors will be highly concentrated around zero, and the larger errors will have smaller probability. This helps in reducing the sparsity of the statistics, as the majority of the residuals will lie in a small subrange of the prediction error alphabet.

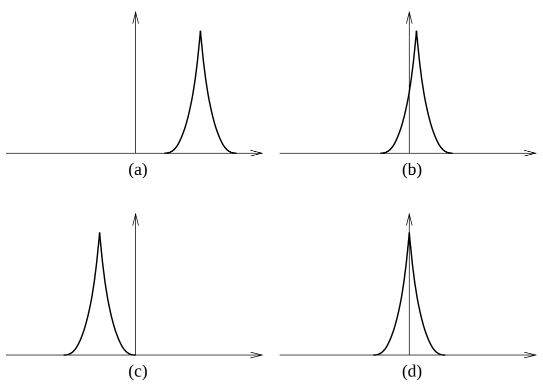


Figure 5.4: Bias cancellation. (a), (b) and (c) are three distributions of prediction residuals with the same shape but centered at different places (biases). By removing these biases, the three distributions can now be merged in only one.

From the work in [19] it has been an accepted fact that prediction error distributions often obey a Two-Sided Geometric Distribution (TSGD) centered around 0. As mentioned in [17] and [32], when the prediction error distribution is also considered context-dependent, the resulting distribution of each context is still TSGD-like but centered around a context-dependent *bias*. The TSGD is defined as

$$P_{X|\gamma}(x) = \frac{1-\theta}{\theta^{1-s}+\theta^{s}}\theta^{-|x-\mu|} \tag{5.1}$$

where $\theta$ (decay term) and $\mu$ (the center, which corresponds to the mean of the distribution) are parameters of the distribution and $s = \lceil \mu \rceil - \mu$ is a term between 0 and 1.

Suppose now that there are many contexts in which the prediction error has approximately the same shape, but centered at different offsets depending on each context. If those shapes correspond to the same distribution, they are centered around 0 and merged to obtain a better estimation of the distribution. This is illustrated in Figure 5.4.

This gives rise to a special case of context-dependent prediction called *bias cancellation*. In this scheme, used in many succesful compression tools such as LOCO-I [33], the predictor consists of a fixed part and a context-dependent adaptive *bias* term that is used to center the prediction residual distribution around 0.

When working with context-dependent prediction, and for the same reasons (the growth in the number of contexts with the size of the alphabet), the same context classification approach that was used to relate similar contexts in the conditional distribution estimation problem is used. The objective in this case is to adjust the *bias* term of the predictor for each possible context class. To avoid confusion, these classes will be called *prediction conditioning classes.*

Note that the probability conditioning clases, and the prediction conditioning classes *need not be the same.* One approach is to define the latter classes to be a *refinement* of the former classes (i.e., each probability conditioning class is broken into disjoint prediction conditioning classes). This can be justified by the assumption that the classification used for the context-conditional distributions joins contexts in which the prediction errors have the same distribution shape

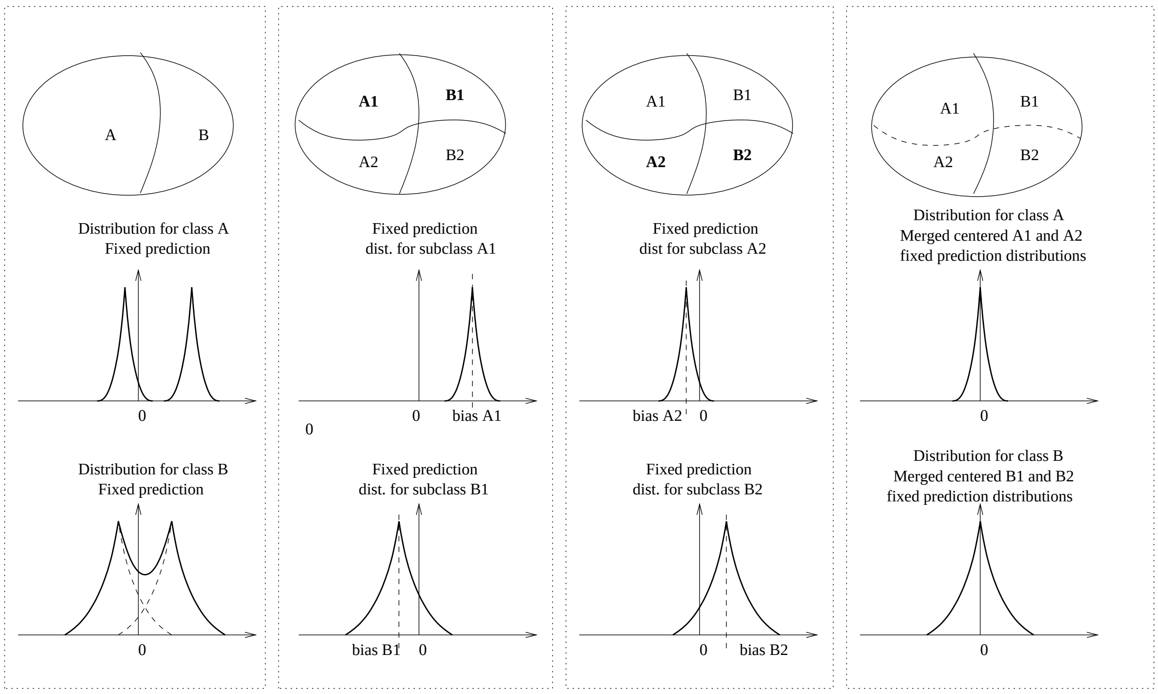


Figure 5.5: Role of context-conditional prediction (bias cancellation).

(same shape and moments except the mean), but have their center (mean) at different places; and the classification used for bias cancellation produces a partition of each class in a set of *sub-classes* where the predictor bias is the same for all the contexts in the same sub-class. This is the kind of scheme used in the LOCO-I algorithm and is exemplified in Figure 5.5.

## 5.3 Prediction and denoising

Assume that the context class, the noisy value and the prediction for the current noisy symbol are $\gamma_i \in \mathcal{A}^K$, $\mathbf{z}_i$ and $\hat{\mathbf{z}}_i$ respectively. Notice that $\hat{\mathbf{z}}_i$ is the prediction of the **noisy** sample at the center of $\gamma_i$. This might seem counter-intuitive at first, since the exact value of $\mathbf{z}_i$ is known. As mentioned in the previous section, the idea here is to reduce the sparsity of the context-conditional statistics by concentrating them around 0.

The prediction error for $\mathbf{z}_i$ is defined as

$$\mathbf{e}_i = \mathbf{z}_i - \hat{\mathbf{z}}_i \tag{5.2}$$

Let $Z$, $\hat{Z}$ and $E$ be the random variables modeling these three values, and $\gamma$ be a random vector modeling the possible values of the context classes $\gamma_i$.

In the second pass of the DUDE, the denoiser function for the current sample is defined in terms of the empirical distribution of the input alphabet conditioned on the current context. In the augmented framework, the sample is conditioned on the context class $\gamma_i$.

In order for the second pass to work properly in the augmented framework, the prediction error distribution for the noisy samples has to be reinterpreted in terms of the original noisy distribution.

In Section 5.1 it was mentioned that prediction can be seen as a way to merge similar distributions centered at different offsets, where the offsets are given by the predicted value. Thus the prediction error distribution for the current context class $\gamma_i$ and the current prediction $\hat{\mathbf{z}}_i$

$$P(E = \mathbf{z}_i - \hat{\mathbf{z}}_i | \gamma = \gamma_i, \hat{Z} = \hat{\mathbf{z}}_i)$$

can be assumed to be a centered version of the original noisy distribution when conditioned on the prediction value

$$P(Z = \mathbf{z}_i | \gamma = \gamma_i, \hat{Z} = \hat{\mathbf{z}}_i)$$

that is

$$P(Z = \mathbf{z}_i | \gamma = \gamma_i, \hat{Z} = \hat{\mathbf{z}}_i) = P(E = \mathbf{z}_i - \hat{\mathbf{z}}_i | \gamma = \gamma_i, \hat{Z} = \hat{\mathbf{z}}_i) \tag{5.3}$$

In principle, the current prediction $\hat{\mathbf{z}}_i$ could be used as an additional element to characterize the current context besides the conditioning class. This would increase the potential number of conditioning classes from $|\Gamma|$ (the number of context clusters) to $|\Gamma| \times |\mathcal{A}|$ (since there are $\mathcal{A}$ possible prediction values). In this framework this option will not be considered, assuming that the prediction error distributions are independent of the actual predicted value $\hat{\mathbf{z}}_i$,

$$P(E = \mathbf{e}_i | \gamma = \gamma_i, \hat{Z} = \hat{\mathbf{z}}_i) = P(E = \mathbf{e}_i | \gamma = \gamma_i). \tag{5.4}$$

Sllowing all the statistics of class $\gamma$ to be gathered in one vector $m_\gamma$.

With this assumption and (5.3), the estimated noisy conditional distribution for the current sample $\mathbf{z}_i$ will be

$$P(Z = \mathbf{z}_i | \gamma = \gamma_i) = P(E = \mathbf{e}_i | \gamma = \gamma_i).$$

## 5.4 Prefiltering

The modeling tools that were mentioned in the previous sections (context classification and prediction) assume a certain degree of smoothness in the images to be denoised. To give an example, one of the tools used for grouping contexts is Vector Quantization [15] which joins contexts that are close in terms of their Euclidean distance in context space. Another example is to use the context average value (the average of the context samples) to predict the center pixel. If the image is corrupted by additive noise of relatively small variance (low SNR), these tools will still work, as the contexts which were originally near in the clean image will still be close in the noisy image (since they are vectors of slighty displaced samples). However, if the noise is not additive (such as the “Salt & Pepper” noise), the smoothness assumption will not hold and these tools will not work properly.

To address this issue, the augmented framework includes an optional *prefiltering* pass in the algorithm which takes the noisy input image and produces a *prefiltered* image $\mathbf{y}^{m\times n}$ using some denoising filter. When this scheme is applied, the context class $\gamma_i$ and the prediction $\hat{\mathbf{z}}_i$ for each noisy pixel $\mathbf{z}_i$ are computed from the prefiltered context at the same position $i$, $\mathbf{y}(W_{i,T})$ instead of the noisy context $\mathbf{z}(W_{i,T})$. Statistics are still computed with respect to the original noisy values $\mathbf{z}_i$ as in (5.2).

Prefiltering can also be seen as a way to "expand" the effective contextual information when building the contexts, since the samples in each neighborhood of the prefiltered image would include information from samples outside the neighborhood window. For instance, if the prefilter is based on a sliding window (such as the linear filters described in 3), the "effective" neighborhood would grow up to the radius of the window defined by the filter window size.

## 5.5 Noise preclassification

In some cases it is possible to detect or estimate which pixels of the image are corrupted by noise. This makes sense when dealing with non-additive noise such as impulse noise in which not every pixel is corrupted and, when corrupted, its noisy value is always one of 0 or $M-1$ for an alphabet $\mathcal{A} = \{0, 1, \ldots, M\}$. In this case, a simple detection scheme would be to mark each pixel whose value is either 0 or $M-1$ as a *noisy candidate.* Clearly both values can happen in a clean, uncorrupted image, thus resulting in pixels can be marked as noisy when they are not. A *preclassification mask* $\mu^{m\times n}$ is a binary meta-image where a symbol value of 1 means that the pixel is deemed to be noise, and 0 means that it is not. When available, this meta image is a valuable tool for the following stages of the denoising process.

## 5.6 Loss model

In denoising problems such as binary channel denoising or DNA sequencing denoising there is no sense of proximity between the symbols, and the cost incurred is either the same in all the cases (Hamming cost) or dictated by specific rules. In contrast, continuous-tone images have a *distance relationship* between their symbol values, which can be used to define a metric between the noisy and the clean images. In a grayscale image, choosing symbol $a+1$ in place of a correct $a$ is usually unnoticeable when working with 256 levels of gray. Generally speaking, bigger differences (errors) are more visible than smaller ones.

Because of this, and also because they yield very fast closed form solutions for the argument-dependent minimization used in the denoiser function (4.4) (see Section 6.7 and Appendix B for details), two loss models for continuous-tone images are used:

**absolute difference** Setting each element of $((\Lambda))$ as $\Lambda_{ij} = |i-j|$ an $L_1$ norm is stablished as the distance between the noisy and the clean image. Thus we will refer to this loss model as $L_1$.

**quadratic difference** Here $\Lambda_{ij} = (i-j)^2$ and the associated distance corresponds to the square $L_2$ norm between the noisy and the clean image. This will be referred to as the $L_2$ loss model.

# 6 Proposed solution

In this section the details of the augmented DUDE framework for continuous-tone images, or *DUDE-I* for short, are described.

## 6.1 Description of the framework

**The block diagram** for the DUDE-I is depicted in Figure 6.1.

**The Prefilter** takes the noisy image $\mathbf{z}^{m\times n}$ as input and produces a prefiltered version of it, $\mathbf{y}^{m\times n}$, that can then be used by the Modeler for context extraction and prediction.

**The Preclassifier** , when used, computes a binary mask $\mu^{m\times n}$ where $\mu_i = 1$ for those $\mathbf{z}_i$ that are deemed to be corrupted noise and $\mu_i = 0$ otherwise.

**The Modeler** classifies each sample of the image $\mathbf{z}_i$ into a context class $\gamma_i$, producing a meta-image $\gamma^{m\times n}$ named *conditioning map.* The number and characteristics of each class is defined by the Modeler and may vary with the actual data.

Along with the conditioning map, the Modeler also produces an optional prediction $\hat{\mathbf{z}}^{m\times n}$ of the image which can be used to further simplify the probabilistic model of the image.

**The Denoiser** depends on the **Channel Model** and the **Loss Model** to select between different strategies that are suitable for each case. For instance, the second pass of the DUDE-I for the Gaussian Noise contains special steps and subalgorithms not found in the corresponding second pass for the Impulse Noise (and its variants).

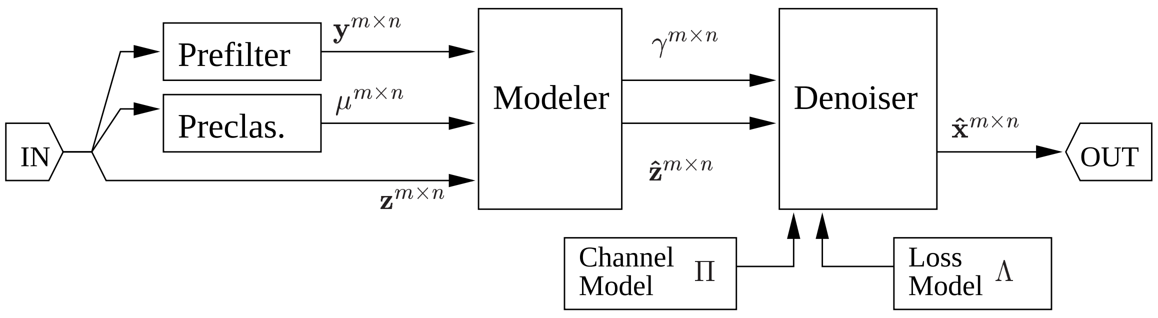


Figure 6.1: Block diagram for the DUDE-I.

## 6.2 Preclassification schemes

### 6.2.1 Trivial S&P detection scheme

This is the most straightforward prefiltering scheme to apply when confronted to an image corrupted by impulse noise. As the only possible noisy values are 0 and $M-1$, this algorithm marks all those pixels of $\mathbf{z}^{m\times n}$ with a those values as noisy. Despite its simplicity, this simple approach improves the overall performance significantly compared to the case where no preclassification is done.

$$\mu_i = \begin{cases} 0 & , \quad 0 < \mathbf{z}_i < M-1 \\ 1 & , \quad \text{otherwise} \end{cases} \tag{6.1}$$

### 6.2.2 Thresholding

This is a variant of the preceding algorithm where the symbols are marked as noisy if their values are a certain threshold $\tau$ appart from the extreme values 0 and $M-1$:

$$\mu_i = \begin{cases} 0 & , \quad \tau < \mathbf{z}_i < M-1-\tau \\ 1 & , \quad \text{otherwise} \end{cases} \tag{6.2}$$

### 6.2.3 Binary DUDE

The Impulse channel, as described in Section 2.2.2, is not exactly the same as an Erasure channel, since the erased symbols take *valid* input alphabet values (0 for pepper and $M-1$ for salt) instead of a special *erasure* value that is added to the output alphabet.

This fact motivated approaches such as [24], where the main goal is to determine which of the symbols of the output having an erasure value are actually noisy symbols (and thus should be replaced), or clean symbols that happen to have one of those unfortunate values.

The DUDE-I does not change any symbol which has *not* the erasure value, but it may change symbols that have it.

This scheme uses a binary DUDE similar to the one used in [21] to produce the actual preclassification. For simplicity, consider the Z-Channel with probability of error $\lambda$. Consider a sequence $\mathbf{x}^{m\times n}$ that has been corrupted by this noise yielding a noisy sequence $\mathbf{z}^{m\times n}$. The *erasure symbol* of the Z-Channel has a value $e \in \mathcal{A}$. Now take the *noise mask* meta-image as produced by the trivial preclassification scheme described earlier, $\mu^{m\times n}$. This meta-image will be called $\mu_z^{m\times n}$. This is a binary meta-image where $\mu_i = 1$ indicates that $\mathbf{z}_i$ is a *potential* noisy sample.

The key is to consider the $\mu_z^{m\times n}$ meta-image as a *noisy binary image* itself. For this, consider the noise mask which would be obtained from the (unobserved) clean image $\mathbf{x}^{m\times n}$ by the trivial scheme. This meta-image will be called $\mu_x^{m\times n}$.

This is also a binary sequence but now it marks those clean pixels that coincide with the erasure symbol $e$. If a pixel in the clean mask was 1, then it can only be 1 in the noisy mask because it would also be $e$. If it was 0, however, it has a probability of exactly $\lambda$ of becoming 1. Thus, for each index $i$,

$$
\begin{array}{l}
P(\mu_z[i] = 0|\mu_x[i] = 1) = 0 \\
P(\mu_z[i] = 1|\mu_x[i] = 1) = 1 \\
P(\mu_z[i] = 0|\mu_x[i] = 0) = 1 - \lambda \\
P(\mu_z[i] = 1|\mu_x[i] = 0) = \lambda
\end{array}
\tag{6.3}
$$

This itself corresponds to the behavior of the binary Z Channel and its transition matrix is, according to (6.3),

$$
\Pi = \begin{vmatrix} 1-\lambda & \lambda \\ 0 & 1 \end{vmatrix}. \tag{6.4}
$$

The binary DUDE for this channel can be applied to obtain a *denoised mask* $\mu_{\hat{x}}^{m\times n}$ from the noisy mask $\mu_z^{m\times n}$.

The denoised mask will keep those pixels that coincide with the erasure value but are *not* noisy. However, the denoised mask is defined to contain only the noisy pixels. To obtain the final noise mask $\mathbf{m}u^{m\times n}$ observe that

- $\mu_z[i] = 1$ indicates either a false or a true noise detection.
- $\mu_{\hat{x}}[i] = 1$ indicates (ideally) only a false detection.

Thus, the $i$-th symbol of the desired noise mask, $\mu_i$, will be 1 if $\mu_z[i]$ is 1 but $\mu_{\hat{x}}[i]$ is 0. This can be expressed as a logical symbol-wise operation between the two masks:

$$
\mu^{m\times n} = \overline{\mu}_{\hat{x}}^{m\times n} \wedge \mu_z^{m\times n}
$$

where $\overline{\mathbf{a}}$ indicates the bitwise *negation* of $\mathbf{a}$ operation and $\wedge$ the bitwise *and* operation.

To obtain a mask for a multivalued erasure-like channel such as the Salt and Pepper, the scheme is easily extended using a $q$-ary Z-Channel or by obtaining separate masks for each erasure value using the previous scheme, and combining them with a bitwise *or* operation between the masks.

### 6.2.4 Discrimination by homogeneity level

This scheme, which was described in [24] for the detection of Salt & Pepper noise, can also be used with more difficult non-additive noise models such as the $q$-ary symmetric channel. The basic idea is to mark pixels as noisy when their values are not likely to occur given their context.

To do this, the *co-occurence matrix* [9, pp. 416–417] of the noisy image is computed. This tool has been given many interpretations and variants in the literature, usually using the same name. The approach followed is that of [24] where the co-occurrence matrix is an $M \times M$ matrix $H = \{h_{ij}\}$ where $h_{ij}$ corresponds to the number of times the symbol $j$ occured in a $3 \times 3$ context whose center symbol is $i$ (denoted by $C_{i,3\times3}$ with $|C_{i,3\times3}| = 8$) all over the image:

$$
h_{ij} = \sum_{i \in R_{m\times n}} \sum_{j=0}^{8} C_{i,3\times3}[k] \tag{6.5}
$$

Each row $r$ of $H$ can be seen as a a histogram of the context samples conditioned on the event that the center sample has a value of $r$. The basic idea is to use these histograms as conditional distributions of the context samples in order to detect outliers. After this matrix is obtained from the noisy image, for each value $r \in \mathcal{A}$ an upper $U_r$ and lower $L_r$ bound for the values that each context sample can take in order for it to be *homogeneous* with $r$ is computed. Assuming that each histogram is monomodal and centered at the center symbol value $r$, the upper and lower bounds are searched as those columns where the histogram values fall below a given threshold $\tau$. More precisely, the distribution at the $j$ column of row $r$ is estimated as an average in a window of size 3 centered around $j$:

$$L_r = \arg\min_j \left\{ h_{rj} : \sum_{k=j-1}^{k=j+1} h_{rk} \geq \tau \right\}$$

$$U_r = \arg\max_j \left\{ h_{rj} : \sum_{k=j-1}^{k=j+1} h_{rk} \geq \tau \right\}$$

Let $\mathcal{H}_{i,3\times3} = \{c \in C_{i,3\times3} : L_{\mathbf{z}_i} \leq c \leq U_{\mathbf{z}_i}\}$ be the set of context samples homogeneous with the center sample at position $i$ for a $3 \times 3$ square context template. With these bounds computed for each symbol $r$, and this definition of $\mathcal{H}_i$, a primary classification $\mu*^{m\times n}$ of the noisy pixels is performed as follows

$$\mu*_i = \begin{cases} 1 & , \quad |\mathcal{H}_{i,3\times3}| > 4 \\ 0 & , \quad \text{otherwise} \end{cases} \tag{6.6}$$

this classification produces an important number of false detections. A refinement pass is then performed using $5 \times 5$ square contexts. In this pass, each pixel initially marked as noise is unmarked if the majority of the context samples marked as homogeneous with it are not marked as noise. This results in the final mask $\mu^{m\times n}$

$$\mu_i = \begin{cases} 0 & , \quad |\{c \in \mathcal{H}_{i,5\times5} : c \text{ clean}\}| > |\mathcal{H}_{i,5\times5}|/2 \\ 1 & , \quad \text{otherwise} \end{cases} \quad , i : \mu^*_i = 1 \tag{6.7}$$

## 6.3 Prefiltering schemes

### 6.3.1 Basic prefiltering

The DUDE-I framework accepts any image filter as a prefilter. The tested prefilters include the classical schemes described in Section 3 such as the Window Median or Window Average. When a preclassification mask is available, the filters are applied only to those pixels marked as noisy.

### 6.3.2 Recursive prefiltering

The prefiltering process can be based on any filter as long as it produces an output that is smoother than the noisy image. If the output of the DUDE-I is indeed closer to the unobserved clean image than the noisy image, its output can used as the prefiltered image used to build the context model in a following stage. This scheme is depicted in Figure 6.2.

**Note** that the noisy input to each denoising pass is always the initial noisy image. *This is not a recursive denoising scheme.*

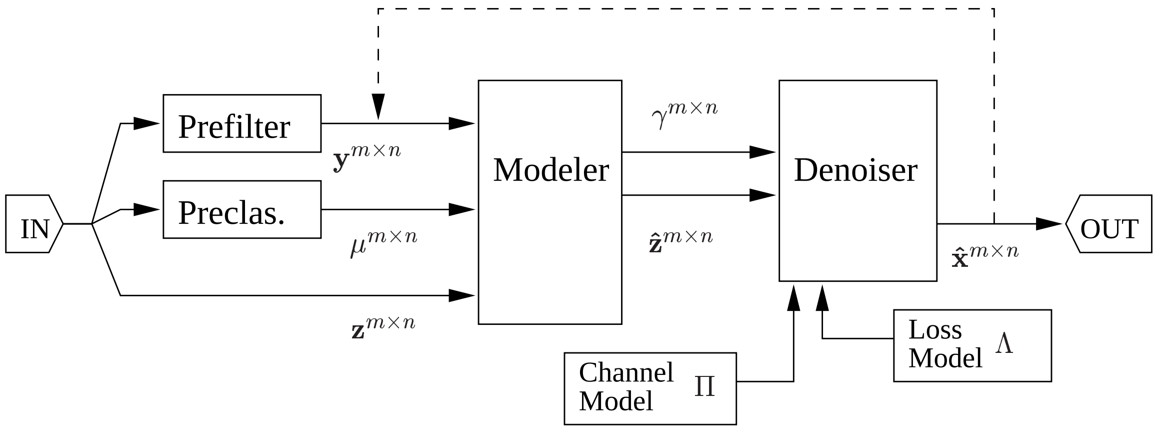


Figure 6.2: Block diagram for the Recursive Prefiltering setting. The loop is closed for the first N-1 cycles and in the N-th cycle the two switches change positions in order to work as in the *normal* configuration of Figure 6.1.

## 6.4 Modeling schemes

The following sections describe the different modeling schemes which were applied in this work.

Being a continuation of the work started by Giovanni Motta [18], the present work inherited some of the tools used in the former. These are referred to as the *Legacy* tools. Of these tools, the *Legacy Modeling scheme* is the first modeling approach to be described here in Section 6.5.

The original work in this thesis is comprised mainly by what the so called *Napkin Modeling Scheme*, described in Section 6.6 below. This scheme was created using the techniques described in Section 5.

## 6.5 The Legacy Modeling Scheme

### 6.5.1 Summary

Given a window size and shape, the Legacy Modeling Scheme gathers all the contexts from the image as vectors, performs a canonical spatial transformation, a DC cancellation of its samples, and and then uses a *vector quantization* (VQ) strategy to classify the resulting contexts into a fixed number of clusters (classes). An *optional* prediction is computed using an arbitrary filter as a predictor.

A block diagram of this modeling approach is depicted in Figure 6.5. Folloging is a detailed description of each stage of the algorithm.

### 6.5.2 Canonical transformation

When gathering the contexts, a transformation is performed to match similar contexts with different orientations (rotation and/or reflection). The idea is to combine a set of four rotations (0,90,180 and 270 degrees) and an optional axial symmetry, so that in the end, the four quadrants of the context are ordered in decreasing intensity. Figure 6.5.2 gives a graphical example of this concept. The algorithm itself is given in Figure 6.3,

- Take the current context as defined by a neighborhood template $T$ and compute the sum of the intensities of its quadrants: $S_{nw}$, $S_{ne}$, $S_{sw}$,$S_{se}$. Here, each quadrant is defined by the relative position of the context sample to the center (to-be-conditioned) pixel. The axes are *not* taken into account, and the context shape (defined by the template $T$) must have central symmetry for the algorithm to work well.
- Rotate the context so that the upper left ($nw$) quadrant has the higher overall intensity $S$.
- If, after the rotation, the lower-left ($sw$) quadrant has more overall intensity than the upper-right ($ne$), then flip the context along the $nw$–$se$ axis so that both quadrants are now swapped.

Figure 6.3: Canonical Transformation algorithm.

### 6.5.3 DC cancellation

Once the context has been canonically transformed, its average sample value is subtracted from the samples that comprise it. This is a way to exploit the similarity between contexts regardless of the local intensity level.

### 6.5.4 Quantization

After all the contexts have been gathered, rotated and their DC has been removed, they are quantized into a fixed number of clusers (which is a key parameter of the algorithm) using the LBG algorithm developed by Linde, Gray and Buzo [15]. The LBG defines the context clusters by finding a set of corresponding cluster centers (one per cluster) in an iterative fashion. The algorithm stops when either there is no further change in the position of the centers on each iteration, or when a maximum number of iterations is reached.

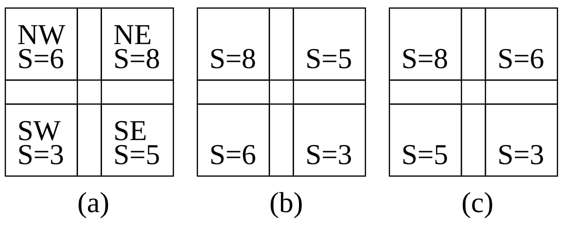


Figure 6.4: Canonical transformation. (a) The four quadrants and their sums. (b) After the rotation, the upper-left quadrant has the largest sum (in this case the rotation was 90 counter-clockwise). (c) Finally, the context is mirrored along the *nw-se* axis so that the upper-right quadrant is brigther than the lower-left one.

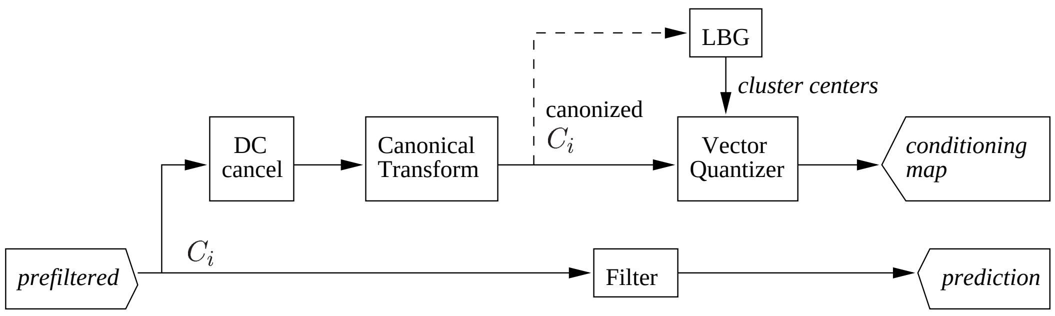


Figure 6.5: Block diagram for the Legacy modeling. The prefiltered noisy sequence is fed to the prediction filter to produce the prediction. The raw contexts from the whole sequence are quantized using the LBG algorithm (dashed line) and then a second pass classifies each context into one of the resulting context classes to form the conditioning class map.

### 6.5.5 Prediction

Prediction is optional and based on an arbitrary filter applied to the noisy or prefiltered image (if prefiltering is used). Common filters such as the *Average* or *Median* filters described in Section 3 were tested in this scheme, but also special ones such as the *Napkin filter* (to be described later in this chapter) were adapted to the Legacy scheme with good results.

## 6.6 The Napkin Modeling Scheme

### 6.6.1 Summary

This algorithm takes the techniques applied in low-complexity image compression algorithms such as [33] and [36], and adapts them to a noisy environment to produce both a context modeling scheme and a prediction scheme that are robust under noisy contexts, and, at the same time, fast so that modern digital images can be processed with practical time and computational requirements.

For instance, a fixed scalar quantization scheme is used to compute the context classes, instead of a vector quantization scheme. Prediction is inspired on the MED predictor used in JPEG-LS [33], extending it to non-causal contexts.

A general block diagram of this modeling scheme is depicted in Figure 6.6. Each block is now described in detail.

### 6.6.2 The Context Wings

The *Napkin Modeling Scheme* derives its name from the fact that it divides the context window in four *wings*; N,S,E and W as shown in Figure 6.7. It then computes *four* directional gradients; $d_N$, $d_S$, $d_E$ and $d_W$ according to Equations (6.8) to (6.11) which use the local differences between the samples at each wing. Each sample within the context is referred to as $c_{xx}$ where $xx = n, s, w, e, nw, ne, \ldots$ indicates the relative position of the sample in the context with respect to the center sample (for instance, $c_{nn}$ indicates the sample which lies to the northwest, i.e., at relative position $(-1, -1)$).

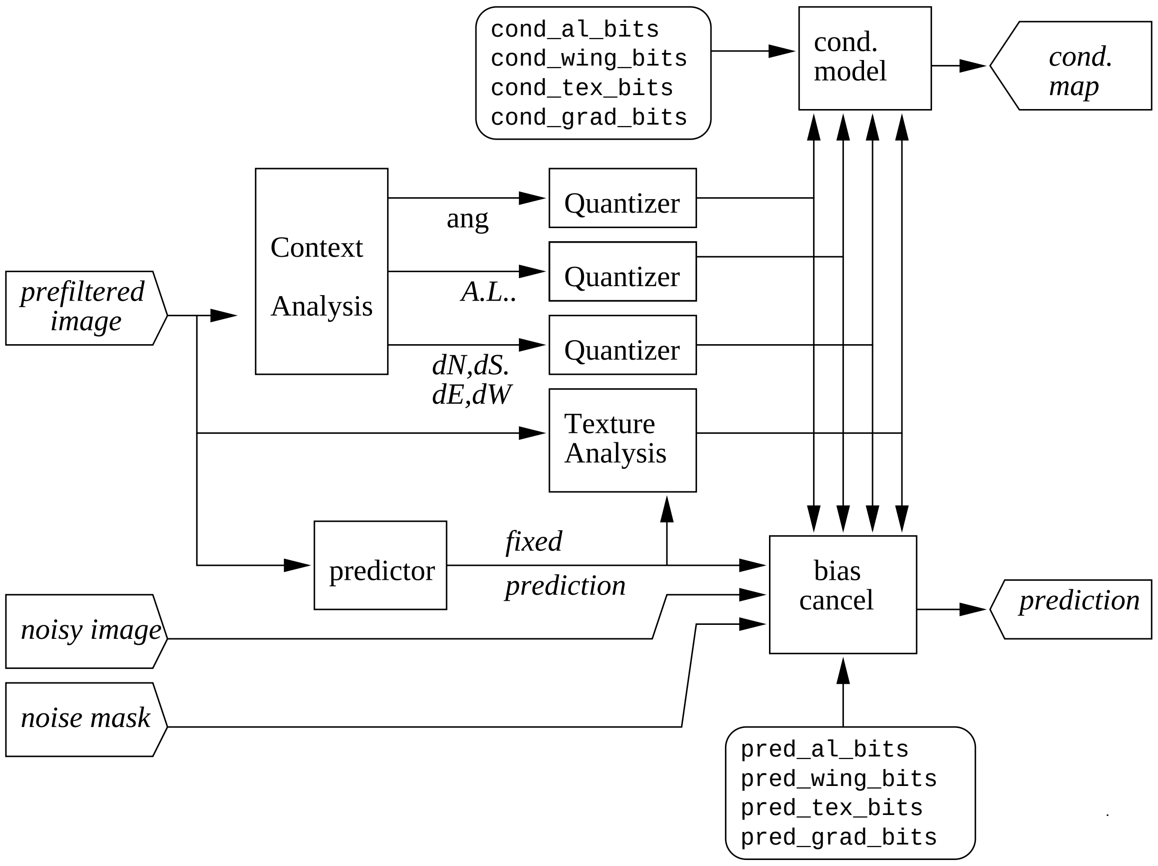


Figure 6.6: Block diagram for the Napkin modeling.

$$\begin{aligned}
d_N &= c_n - c_{nn} + c_e - c_{ne} + c_w - c_{nw} && (6.8)\\
d_S &= c_{ss} - c_s + c_{se} - c_e + c_{sw} - c_w && (6.9)\\
d_E &= c_{ee} - c_e + c_{ne} - c_n + c_{se} - c_s && (6.10)\\
d_W &= c_w - c_{ww} + c_n - c_{nw} + c_s - c_{sw}. && (6.11)
\end{aligned}$$

This directional gradient information is then used both to to determine the way in which the center sample value will be predicted, and the context class to which the sample belongs.

Each gradient is a *signed* sum of three adjacent local gradients in the same direction. This is a tradeoff between locallity of the gradient and noise resilience, because a signed sum will tend to reduce the relative influence of white noise since it acts as a low pass filter. In contrast, in the compression applications that have been mentioned many differences are added in terms of their *absolute* values.

**Broad variant** If the noise is additive and its power is high, the average of three local differences may not be enough to reduce its influence. Because of this, a *Broad Variant* exists which computes each wing gradient using five samples. In this case the gradients are obtained

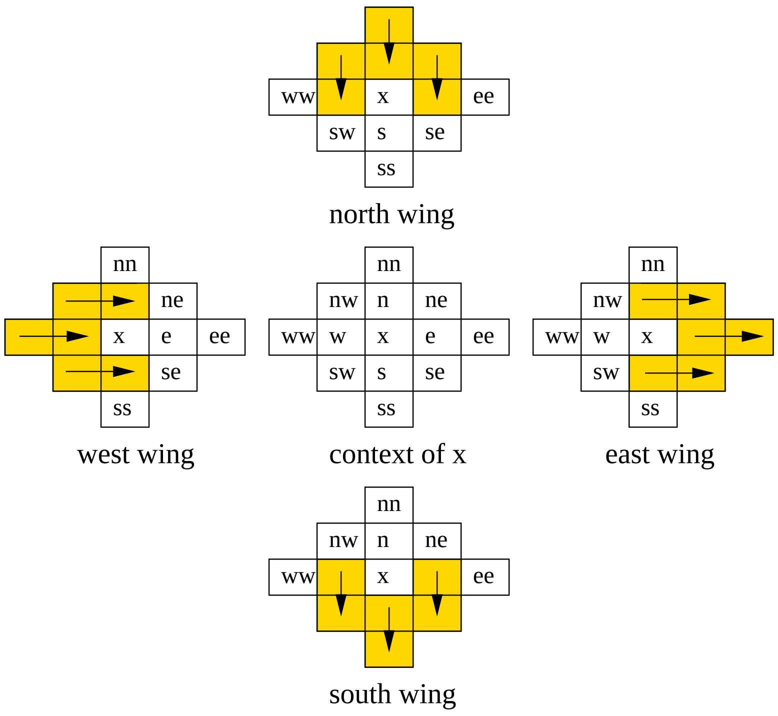


Figure 6.7: Wing gradient computation. The 12-pixel diamond-shaped context at the center is broken into four (overlapping) wings. For each wing, a gradient is computed as the average of three local differences. The small arrows show the direction and the samples involved in each local difference computation for each wing.

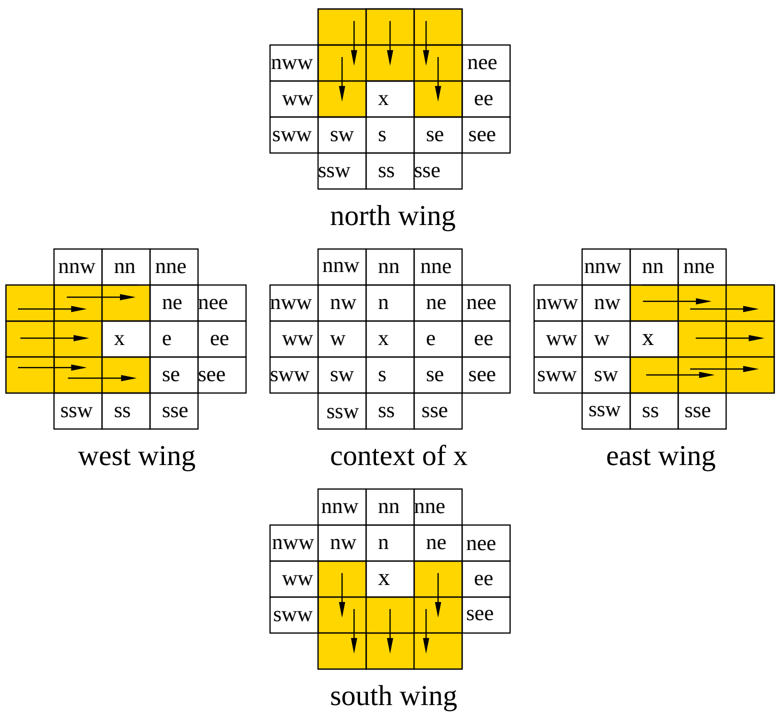


Figure 6.8: Wing gradient computation for the Broad variant. Eight additional samples are required: $nnw, nne, ssw, sse, see, nee, sww$ and $nww$.

using Equations (6.12)—(6.15). The context and the wings used in this variant are depicted in Figure 6.8.

$$
\begin{aligned}
d_N &= c_n - c_{nn} + c_e - c_{ne} + c_w - c_{nw} + c_{ne} - c_{nne} + c_{nw} - c_{nnw} && (6.12)\\
d_S &= c_{ss} - c_s + c_{se} - c_e + c_{sw} - c_w + c_{sse} - c_{se} + c_{ssw} - c_{sw} && (6.13)\\
d_E &= c_{ee} - c_e + c_{ne} - c_n + c_{se} - c_s + c_{nee} - c_{ne} + c_{see} - c_{se} && (6.14)\\
d_W &= c_w - c_{ww} + c_n - c_{nw} + c_s - c_{sw} + c_{nw} - c_{nww} + c_{sw} - c_{sww}. && (6.15)
\end{aligned}
$$

The four directional gradients are combined into two *orientation* gradients $d_H$ and $d_V$ in *absolute* terms,

$$
\begin{aligned}
d_H &= |d_E| + |d_W| && (6.16)\\
d_V &= |d_N| + |d_S|. && (6.17)
\end{aligned}
$$

Finally, an overall activity level is also computed from these two gradients,

$$AL = d_H + d_V. \qquad (6.18)$$

The reason for having such hierarchy is to be able to recombine them so that different tradeoffs can be selected in terms of precision in the characterization of the region and noise resilience.

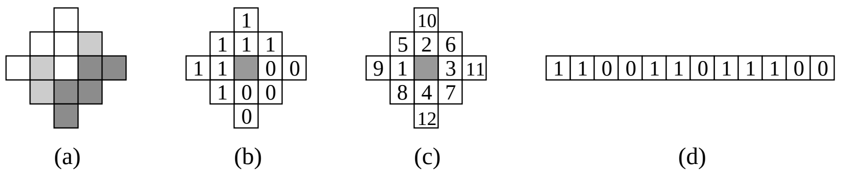


Figure 6.9: Texture bitmap computation: The predicted value is compared to the raw context samples producing either a 1 (above prediction) or a 0 (below prediction) for each sample. (a) shows a given context, (b) shows the result of the comparison, (c) shows the order of the samples in the context vector and (d) the resulting texture bitmap.

## 6.6.3 Context Modeling

In the current classification-based context modeling framework, the modelers aim at producing a minimal set of characteristics for which the contexts that fall in a same group (class) are similar in a way useful to the system, i.e., share similar empirical probabilities of the noisy center sample conditioned on the noisy contexts.

As a classification problem, the goal is to find these optimal characteristics. With model cost [26] added to the problem, the optimal set of characteristics stems from a tradeoff between context description power and the possible number of contexts. If noise is taken into account, sensitivity in the measures of these characteristics is another problem to deal with.

The complexity of this scenario led to the development of a flexible scheme for the selection of these characteristics. The result is that context classes can be formed from the combination of three measures: *quantized activity level*, *quantized wing gradients* and *texture bitmap*. Each of these measured characteristics represent a different tradeoff between precision and expresiveness.

### Activity Level (AL)

This measure represents a global activity level of the region spanned by the current window context.

Being a global magnitude that results from the combination of many other measures, this measure should be the least affected by noise from the three, while its ability to characterize a context is limited to its global nature (no hint of spatial structure can be derived from it).

### Texture Bitmap

The texture bitmap tries to capture a basic texture pattern from the context. In contrast to the Activity Level, it is highly expressive but also highly sensitive to noise. It can be used in conjunction with the other features to regain some of the structural information that they do not capture.

To compute the texture bitmap, each pixel in the window is compared to the predicted value. A single bit is used per pixel to indicate if its value was above or equal (1) or below (0) the predicted value. Finally, the bitmap is unrolled into a binary word by traversing the bitmap in a spiral fashion, i.e., as concentric circles of increasing radius. Figure 6.9 shows this procedure

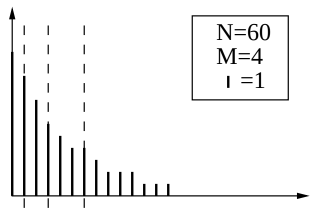


Figure 6.10: Example binning for the quantization scheme used in the Napkin modeler. The histogram, built from 60 hypothetical non-quantized gradients, is partitioned into 4 *bins* so that each region that correspond to the same quantized value has roughly 15 samples in it.

### Wing gradients

The four wing gradients are independently computed and, in combination, can give useful information not only about the overall activity of the region in but also about the shape of this region. For example, if the North gradient is positive, and the South gradient is negative, there is a local maximum in the *vertical* direction. If, at the same time, both the East and West wings gradients have the same sign or are flat,then there is continuity in that direction, a situation that could arise if a line of the image is traversing the context.

This scheme is a tradeoff between the two previous features since it gives a better description of the contextual structure than the activity level by itself, while being more robust to noise than the binary texture component.

### Gradient direction

Another descriptive element which proved to be useful is the estimation of the direction of the overall context gradient. This is computed as

$$\phi = \tan^{-1}\left(\frac{d_N}{d_S}\right). \tag{6.19}$$

### Quantization

To quantize the magnitudes involved in the context modeling (activity level, wing gradients and gradient direction), a non-uniform quantization algoritm was developed.

The main idea of this algorithm is to produce a quantization in which the resulting quantized values yield a uniform distribution (thus having maximum empirical entropy), i.e., so that each quantized level has roughly the same occurence within the image.

To achieve this goal, the algorithm takes the histogram of the magnitude to quantize as it appears for the whole image and breaks it into regions (also called *bins*) which have roughly the same number of samples inside of them. This idea is depicted in figure Figure 6.10 for a sample gradient histogram.

The algorithm takes as input the unquantized histogram of the magnitude to be quantized $H$ that goes from 1 to $N$, and a number of bins to where the raw values will be put into, $B$. The histogram is assumed to be a monotonically decreasing function of $1 \leq n \leq N$. An outline of the algorithm is shown in Figure 6.11.

```
· Compute $T = \sum_{n=1}^{N} H[n]$.
· Set $t = N$, the threshold pointer.
· while $t > B$,
    · Compute $M = \lceil T/B \rceil$, the target number of hits per bin.
    · while $A < M$ and $t > B$,
        · add $H[t]$ to $A$.
        · decrement $t$.
    · Set the current value of $t$ as one of the quantization thresholds.
    · Decrement $B$.
    · if the number of remaining levels $t = B$,
        · Assign all the remaining levels as thresholds, yielding $t$ one-level quantization bins.
        · END.
    · Update $T = T - M$
    · Update $M = \lceil T/B \rceil$
    · Set $A = 0$
```

Figure 6.11: Maximum entropy binning algorithm.

Pathological situations (such as $B = 0$) are ommited for the sake of clarity.

### Conditioning class computation

When the Activity Level, the Texture, gradient Direction and the Wing Gradients have been computed, a conditioning class is defined for the current pixel which combines the four features into a unique numerical *signature* by concatenating their binary representations (Figure 6.12).

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $a_1$ | $\cdots$ | $a_{na}$ | $t_1$ | $\cdots$ | $t_{nt}$ | $\phi_1$ | $\cdots$ | $\phi_{n\phi}$ | $w_1$ | $\cdots$ | $w_{nw}$ |

Figure 6.12: Conditioning class computation. $na$ stands for number of activity level bits, $nt$ for texture bits, $n\phi$ for gradient direction bits and $nw$ for wing gradient bits.

## 6.6.4 Prediction

Prediction in the Napkin Modeler was broken into a fixed predictor term and a context-dependant variable term (bias cancellation, whose general description was given in Section 5.2). This is similar to the approach used in LOCO-I [33].

In a first pass, a fixed prediction is computed for the whole image and a complementary context model, the "bias cancellation model", is used to perform a context dependant bias cancellation.

For the fixed part of the predictor, a baseline algorithm was developed, called *Average Napkin*, along with two variants: the *Sharp Napkin* and the *Smooth Napkin*, which are tailored for the two main types of noise studied (non additive and additive respectively).

The basic idea of the three variants is to predict the center sample using only those samples from its surrounding window which are smooth, and not part of rapidly changing regions (edges, lines, etc.). To measure the smoothness of each region, the *wing gradients* that were described earlier are used to produce a *wing weight* proportional to the smoothness of the region. These weights are defined as

$$\begin{aligned} w_N &= 1/(1+|d_N|) && (6.20)\\ w_S &= 1/(1+|d_S|) && (6.21)\\ w_E &= 1/(1+|d_E|) && (6.22)\\ w_W &= 1/(1+|d_W|). && (6.23)\end{aligned}$$

## 6.6.5 Fixed prediction variants

**The Average Variant** computes a per-wing average and then produces a prediction using only the averages from those wings that are deemed to be *flat*, i.e., whose sample values do not vary more than a certain amount. The idea is to predict the center sample using only those samples whose values are deemed to be close to its (unknown) value.

The *flatness* criterion is based on the relative magnitudes of the four wing gradients. First, the minimum wing gradient magnitude is computed,

$$d_m = min\left(|d_N|, |d_E|, |d_S|, |d_W|\right). \tag{6.24}$$

A wing is considered to be flat if its gradient magnitude is no greater than $d_m$ by a fixed threshold, $\theta$, defined as

$$\theta = \texttt{grad\_thres} \times 3 \times |\mathcal{A}| \tag{6.25}$$

or, if the broad variant is used,

$$\theta = \texttt{grad\_thres} \times 5 \times |\mathcal{A}| \tag{6.26}$$

where $|\mathcal{A}|$ is the alphabet size and $0 \leq \texttt{grad\_thres} \leq 1$ is a parameter of the algorithm.[1] To produce a final result using only the flat wings, a second set of weights is obtained,

$$w'_N = \begin{cases} w_N & , \quad |d_N| - d_m < \theta \\ 0 & , \quad otherwise \end{cases} \tag{6.27}$$

$$w'_S = \begin{cases} w_S & , \quad |d_S| - d_m < \theta \\ 0 & , \quad otherwise \end{cases} \tag{6.28}$$

$$w'_E = \begin{cases} w_E & , \quad |d_E| - d_m < \theta \\ 0 & , \quad otherwise \end{cases} \tag{6.29}$$

$$w'_W = \begin{cases} w_W & , \quad |d_W| - d_m < \theta \\ 0 & , \quad otherwise \end{cases} . \tag{6.30}$$

[1]The maximum possible gradient is three times the alphabet size because it is the signed sum of three local differences which can only differ in $|\mathcal{A}|$. The case of the broad variant is analogous.

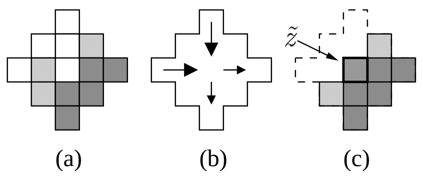


Figure 6.13: Average Napkin variant. (a) raw context. (b) wing gradients. (c) the fixed prediction $\tilde{\mathbf{z}}$ is computed as a weighted average of the flat wings.

The wing averages are computed as follows:

$$a_N = (c_n + c_{nn} + (c_{nw} + c_{ne})/2)\,/3 \tag{6.31}$$
$$a_E = (c_e + c_{ee} + (c_{ne} + c_{se})/2)\,/3 \tag{6.32}$$
$$a_S = (c_s + c_{ss} + (c_{sw} + c_{se})/2)\,/3 \tag{6.33}$$
$$a_W = (c_w + c_{ww} + (c_{nw} + c_{sw})/2)\,/3. \tag{6.34}$$

The $ne$,$nw$,$sw$ and $se$ context samples are divided by two because wings overlap at those positions. Finally, with all these values calculated, the fixed prediction produced by the Average Napkin is

$$\hat{z} = \frac{w'_N a_N + w'_S a_S + w'_E a_E + w'_W a_W}{w'_N + w'_S + w'_E + w'_W} \tag{6.35}$$

A graphical scheme of this prediction is shown in Figure 6.13.

**The Sharp Variant** differs from the Average Variant in that the weights of the samples of the context can take only two possible values: 0 or 1. The prediction is then computed using only those samples whose weight is 1. The weights are defined as follows: first,a *wing gradient sign* is computed for each wing as,

$$s_x = \begin{cases} -1 & , \; d_x < -\texttt{grad\_thres} \\ +1 & , \; d_x > \texttt{grad\_thres} \\ 0 & , \; otherwise \end{cases} \tag{6.36}$$

where $x$ is one of $N, S, W, E$. In the Sharp Variant, a wing is said to be flat only if its corresponding wing gradient sign $s_x$ is equal to 0. Depending on the flatness of the four wings, the algorithm switches between two modes of operation: *flat mode* or *nonflat mode.*

If *any* of the wings is classified as flat, the predictor works in flat mode. In this mode, all the samples belonging exclusively to flat wings will be used in the prediction (i.e., will have weight 1). This discards samples that overlap two wings and one of them is not flat. The result is a plain (i.e., non-weighted) average of the selected samples which can be written as,

$$\hat{z} = \frac{\sum_{c\in C} w_c c}{\sum_{c \in C} w_c} \tag{6.37}$$

where $C$ is the current context, $c$ are the context samples and $w_c$ is 1 if $c$ belongs only to flat wings and 0 otherwise.

If none of the wings is classified as flat, the predictor switches to the nonflat mode. In this ase, the region is characterized into either a *ridge*, or a *saddle* by looking at the relationship between the two gradients of each main direction (horizontal or vertical). In the first case, the gradient signs match in one direction and are opposite in the other. This would represent a ridge since there is continuity in one direction and a local maximum/minimum in the other, and thus it would be appropiate to use only the samples aligned with the continuous direction (dimension) to predict. In the other case, there can be no distinction between the importance of the wings, but as the gradients are high in all directions it seems clear that farther samples would not improve the prediction so only the four closest neighbors are used to compute the result. An example of these ideas is depicted in Figure 6.14.

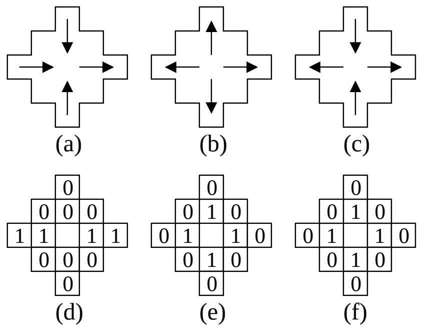


Figure 6.14: Sharp Napkin variant. (a) is a ridge, (b) is a local minimum and (c) is a saddle point. (d), (e) and (f) are the corresponding weights for each case.

**The Smooth Napkin** , in contrast to the previous variants, produces the prediction using *all* the window samples (which can include more samples than the ones used in the wing gradient computation). It assigns a weight to each sample using a per-sample gradient estimation (in contrast to a per-wing approach) and the relative distance of the sample to the center sample to be predicted. This idea is inspired on the anisotropic filter described in Section 3.7.3.

If $c_j \in C$ is a sample of the current context $C$ with relative position $j \in \mathbb{Z}^2$ (for example the *nne* sample has relative position $j = (-2, 1)$), its associated weight is defined as

$$w_j = \begin{cases} j_0 \geq 0, j_1 \geq 0 & , \quad (w_S j_0 + w_E j_1)/|j|^2 \\ j_0 \geq, j_1 < 0 & , \quad (w_S j_0 - w_W j_1)/|j|^2 \\ j_0 < 0, j_1 \geq 0 & , \quad (-w_N j_0 + d_E j_1)/|j|^2 \\ j_0 < 0, j_1 < 0 & , \quad (-w_N j_0 - w_E j_1)/|j|^2 \end{cases} \tag{6.38}$$

where $w_N$, $w_N$, $w_N$, $w_N$ are defined in equations (6.20) through (6.23) This results from considering one weight vector per quadrant (NE,NW,SE,SW) and having each sample weighted by the inner product of its relative position with the vector that corresponds to its quadrant, normalized and then divided by its distance from the center (thus dividing by $|j|$ twice). The idea is depicted in Figure 6.15. With all the weights calculated, the prediction is just the weighted average of the samples,

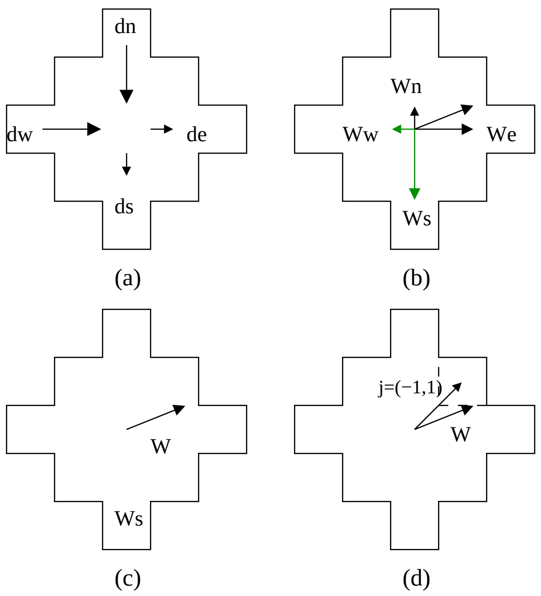


Figure 6.15: Smooth Napkin variant. (a) The wing gradients are computed, (b) their corresponding weight vectors are derived from them, in (c) the weight vector for the quadrant of $j = (-1, 1)$ is computed and (d) shows the relative coordinate vector $j$ and the weight vector $W$. The final weight is the internal product of these two.

### Context dependent prediction: Bias cancellation

The context-dependent part of the predictor consists of a bias cancellation term which is adapted for each possible prediction context class. Following the discussion in Section 5.2, the scheme adopted is that where the context classes used for this adaptive part of the predictor are a refinement of the classes used for building the conditional probability distributions. Thus, the number of bits assigned to describe each of the descriptive components (activity level, wing gradients, gradient orientation, texture) has to be at least the same as those used to build the probability conditioning classes. The prediction conditioning class for a sample $\mathbf{z}_i$ is denoted as $\rho_i$ and all the $\rho_i$ for the image $\mathbf{z}^{m\times n}$ form the *prediction conditioning map* meta-image $\rho^{m\times n}$.

The bias term $b_\rho$ for each prediction class $\rho$ is computed as the average prediction error of the fixed predictor output $\tilde{\mathbf{z}}_i$ and the noisy value to be predicted, $\mathbf{z}_i$ for each sample $\mathbf{z}_i$ that belongs to that prediction class. For this, a *bias counter* is defined which accumulates the differences between $\tilde{\mathbf{z}}_i$ and $\mathbf{z}_i$, and a class counter is incremented to contain the number of occurences of each prediction class. Finally, the bias term is computed as the quotient of both values:

$$b_\rho = \frac{\text{accumulated error for class } \rho}{\text{occurences of class } \rho} \tag{6.39}$$

The bias cancellation techique, being one of the tools used in image compression, is designed to work well with smooth piecewise-constant data such as digital images. Because of this, when the noise is non-additive, the contribution of the noisy samples to the bias term degrades the effectiveness of the technique.

To avoid this problem, the noise mask $\mu^{m\times n}$ produced by the preclassifier is used to exclude those samples marked as noise from the bias term computation. Thus, for a given sample $\mathbf{z}_i$, its fixed prediction $\tilde{\mathbf{z}}_i$ and its prediction class $\rho_i$ the associated bias cancellation term is updated as

$$b_{\rho_i} := \begin{cases} b_{\rho_i} & \text{if } \mu_i = 1 \\ b_{\rho_i} + \mathbf{z}_i - \tilde{\mathbf{z}}_i & \text{if } \mu_i = 0 \end{cases} \tag{6.40}$$

After the first pass is done, the biases are computed by dividing these bias counters by the number of occurences of each prediction conditioning class (which were not deemed to be noisy samples).

With the biases computed, a second pass is performed in which each predicted sample is corrected by the bias which corresponds to its prediction class.

## Algorithm outline

Figure 6.16 summarizes the whole Napkin algorithm for the case of the Average Prediction variant.

```
· Let $\mathbf{z}^{m\times n}$ be the noisy input and $\mathbf{y}^{m\times n}$ its prefiltered version.
· Initialize wing gradient and activity level histograms.
· Initialize the *gradient threshold* using (6.25)
· **First pass**
    · For each $\mathbf{z}_i$ in $\mathbf{z}^{m\times n}$:
        · Extract the context $\mathbf{z}(W_{i,T})$ according to the neighborhood template $T$ (the template definition can be any as long as it includes the wing samples w, n, e, s, nw, ne, se, sw, ww, nn, ee, ss).
        · Compute the wing gradients $d_S, d_N, d_E, d_W$, the directional gradients $d_H$ and $d_V$, the gradient direction $\phi$ and the Activity Level $AL$ using equations (6.8) through (6.18).
        · Add the wing gradient absolute values and the activity level to their respective histograms.
        · Fixed prediction:
            · Compute the minimum wing gradient, $d_m = \min(d_N, d_E, d_S, d_W)$
            · Compute the four wing gradient weights as indicated in (6.20) to (6.23)
            · Compute the fixed prediction, $\tilde{\mathbf{z}}_i$, using (6.31) and (6.35)
        · Obtain the texture bitmap, $\tau$, from the current context and fixed prediction using the method described in Setion 6.6.3.
· Compute the quantization bins for the activity level, wing gradients and gradient direction, based on their corresponding histograms, according to the algorithm described in Figure 6.10. These define three corresponding non-uniform quantization functions $Q_a$,$Q_w$ and $Q_\phi$.
· **Second pass**: classification and bias estimation
    · Initialize the conditioning map $\gamma^{m\times n}$ to hold the probability conditioning class of each pixel in the noisy image. The elements of this map will be referred to as $\gamma_i$.
    · Initialize the *prediction conditioning map*, $\rho^{m\times n}$, to hold the prediction class of each pixel in the noisy image. The elements of this map will be referred to as $\rho_i$.
    · Initialize the *bias* for each *prediction class*, $b_\rho = 0$.
    · Initialize the *counter* for each *prediction class*, $n_\rho = 0$.
    · For each $\mathbf{z}_i$ in $\mathbf{z}^{m\times n}$:
        · Quantize the wing gradients using $Q_w$, the gradient direction using $Q_\phi$ and the activity level using $Q_a$.
        · Compute $\gamma_i$ as a concatenation of the cond_act_bits MSB (most significant bits) of $\Delta$, the cond_tex_bits MSB of $\tau$, the cond_ang_bits MSB of $\phi$, and the cond_wing_bits of $\delta_N$,$\delta_E$, $\delta_S$ and $\delta_W$.
        · Compute $\rho_i$ as a concatenation of the pred_act_bits MSB of $\Delta$, the pred_tex_bits MSB of $\tau$, the pred_ang_bits MSB of $\phi$, and the pred_wing_bits of $\delta_N$, $\delta_E$, $\delta_S$ and $\delta_W$.
        · Update the bias for the current prediction class $b_{\rho_i}$ according to (6.40). If the bias was updated, increment prediction class counter for the current class, $n_{\rho_i}$.
· Third pass: bias cancellation
    · normalize the biases as $b_\rho = b_\rho / n_\rho$ for each prediction class $\rho$.
    · For each $\mathbf{z}_i$ in $\mathbf{z}^{m\times n}$, adjust $\hat{\mathbf{z}}_i = \tilde{\mathbf{z}}_i + b_{\rho_i}$.
```

Figure 6.16: Outline of the Napkin Modeling Scheme.

### 6.6.6 Combined LBG/Napkin

This scheme uses the a simplified version of the Napkin predictor (or any of its variants) as the prediction filter of the Legacy Modeling Scheme. In some cases, this combination has yielded better results than any of the other two modeling approaches. This will be discussed in Section 7.

## 6.7 Denoising Stage

After the probability modeling is defined for the current noisy image, the second pass of the DUDE is performed. There are three variants for doing so, which depend on the selected *loss model*: $L_2$ , $L_1$ or exhaustive search. For the first two cases, fast closed forms of the denoising function (4.4) are available (see Appendix B for a derivation of these closed forms).

For instance, if squared error ($L_2$) is used, (4.4) takes the form of the expectation of the posteriori input distribution $P_{X|C,Z}$:

$$g(\alpha, C) = E\left(\frac{P_{Z|C}\Pi^{-1} \odot \pi_\alpha}{P_{Z|C}(\alpha)}\right) \tag{6.41}$$

here $E(.)$ denotes expectation.

If absolute difference is used, (4.4) corresponds to the median of the posteriori input distribution:

$$g(\alpha, C) = \text{median}\left(\frac{P_{Z|C}\Pi^{-1} \odot \pi_\alpha}{P_{Z|C}(\alpha)}\right) \tag{6.42}$$

where $P_{Z|C}(\alpha)$ term is needed to normalize the resulting vector back to 1 after the element-wise multiplication with $\pi_z$.

Finally, if any other loss model is used, an exhaustive search is done for (4.4) using a *precomputed lookup table* for $\pi_z \odot \lambda_{\hat{x}}$, which is computed only once for each possible combination of $\mathbf{z}_i$ (the current noisy pixel) and $\hat{\mathbf{x}}_i$ (the potential denoiser output).

### 6.7.1 From prediction error to original noisy distribution

For each symbol in the noisy sequence $\mathbf{z}_i$, its corresponding prediction $\hat{\mathbf{z}}_i$ and conditioning class $\gamma_i$ equation (5.3) and the assumption in (5.4) are used to compute each element of $P_{Z|\gamma}$ in terms of $P_{E|\gamma}$ as

$$P_{Z|\gamma}(z) = P_{E|\gamma}(z - \hat{z}) \tag{6.43}$$

The prediction error distribution ranges over an alphabet

$$M' = \{-M + 1, \ldots, -1, 0, 1, \ldots, M - 1\}$$

For a given $\hat{\mathbf{z}}$, there will be elements of $P_{E|\gamma}$ that do not correspond to an element in $P_{Z|\gamma}$. In particular, all those elements of $P_{E|\gamma}$ above $M - 1$, $e \in \{M - \hat{z}, M - \hat{z} + 1, \ldots, M - 1\}$ and below 0, $e \in \{-M + 1, \ldots, -\hat{z} - 1\}$ would be lost. One approach to this problem is to assume that those elements are all mapped to $z = M-1$ and $z = 0$ respectively. Thus (6.43) is modified in the following way:

$$P_{Z|\gamma}(z) = \begin{cases} \sum_{e=-M+1}^{-\hat{z}-1} P_{E|\gamma}(e) & , \quad z = 0 \\ P_{E|\gamma}(z - \hat{z}) & , \quad z = 1, \ldots, M-2 \\ \sum_{e=M-\hat{z}}^{M} P_{E|\gamma}(e) & , \quad z = M-1 \end{cases} \tag{6.44}$$

## 6.7.2 Channel Inversion

**The Channel Inversion problem** which is that of obtaining an estimated input probability distribution $P_{X|\gamma}$ for each context class $\gamma$ requires different strategies for the different types of channels, as the *condition number* [8] of the channel transition matrix, which measures the stability of the solution of the inversion problem, varies greatly depending on the type of channel.

For instance, the Gaussian channel yields transition matrices with very high condition numbers (numerically unstable) for any significant value of its parameter $\sigma$ (*significant* meaning that the noise is actually noticed in the image by visual inspection). In contrast, the non-additive channels yield matrices which do not present numerical problems in their inversion. Furthermore, the inversion can be computed efficiently using closed form solutions for each channel.

Because of this, the problem of Channel Inversion will be discussed for each channel type in subsections Section 6.7.3—Section 6.7.5.

## 6.7.3 Denoising stage for the Gaussian Channel

The transition matrix of the gaussian channel has a high condition number even for small noise variance values (e.g., $\sigma = 1$), which makes the channel inversion problem numerically unstable. Because of this, the channel inversion procedure has to rely on heuristic approaches to obtain input probabilities which capture that part of the information that is still reliable under such conditions.

### The greedy algorithm

Let $\mathbb{P}$ be the set of probability distributions over the alphabet $\mathcal{A}$ and $P_Z$ a channel output probability distribution. When an exact solution can not be found, one possible approach is to obtain an approximation of the input distribution $P_X$, $P^*$, which minimizes the difference between the corresponding approximated output distribution and the true output distribution,

$$P^*{}_X = argmin_{P \in \mathbb{P}} \left( |\Pi^T P - P_Z| \right) \tag{6.45}$$

The greedy algorithm relies on this scheme by doing an exhaustive search on the transition matrix *columns* that generate the output probability subspace. It is defined in Figure 6.17.

The greedy algorithm requires many iterations to converge, which makes it a costly operation. Furthermore, as prediction is performed and so statistics are gathered in terms of prediction errors, the channel inversion has to be computed for each possible combination of context class

```
· set $m = P_Z$
· set $P_X = 0$
· while $|m| > 0$
    · Find the column of $\Pi$ which maximizes the projection of $m$ on it, $i_{max} = \arg\max_{i \in \mathcal{A}} \left( \frac{\langle m^T, \pi_i \rangle}{|\pi_i|} \right)$. $< .,. >$ means dot product and $\pi_i$ is the $i$-th row of $\Pi$.
    · Compute an update term as, $\Delta = \eta \frac{\langle m^T, \pi_{i_{max}} \rangle}{|\pi^i|}$, where $\eta \leq 1$ is used to avoid premature convergence.
    · Update input probability estimation, $P_X(i_{max}) = P_X(i_{max}) + \Delta$.
    · Update the projection residual of $P_Z$, $m = m - \Delta \cdot \pi_{i_{max}}$.
```

Figure 6.17: Pseudocode for the the Greedy Algorithm.

and prediction value. These two facts, when combined, make it impossible to implement any feasible solution for this *channel* without some further simplification.

The simplification used is the one proposed in [18]. This approach assumes that the transition matrix for this channel is *circulant*, i.e., of the form

$$\Pi = \begin{bmatrix} p_0 & p_1 & \cdots & p_{M-2} & p_{M-1} \\ p_{M-1} & p_0 & \cdots & p_{M-3} & p_{M-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ p_1 & p_2 & \cdots & p_{M-1} & p_0 \end{bmatrix}.$$

Given a probability distribution $P_X$ over the alphabet $\mathcal{A} = \{0, \ldots, M-1\}$, its *circular shift of magnitude* $a$ is defined as

$$P_{(a+X)_M}(x) = P_X((x+a)_M)$$

where $(.)_M$ denotes modulo $M$ arithmetic. Under the circulant matrix assumption it can be shown that,

$$\Pi^T P_{(a+X)_M} = P_{(a+Z)_M}, \forall a \in \mathcal{A} \qquad (6.46)$$

Equation (6.46) means that the circular shift and inversion operations are interchangeable.

Until now, when denoising using prediction error statistics, those statistics had to be shifted by the value $\hat{z}$ *before* inverting the channel for each possible $\hat{z}$. As each reconstructed distribution yields different input distributions, the inversion process had to be carried out for each possible combination of prediction value and conditioning class.

Using the circulant matrix assumption, shift and inversion are interchangeable and the channel can be inverted only once for each conditioning class in terms of the prediction error distribution estimated directly from the prediction error statistics, and leave the shifting as the only per-pixel prediction-dependent operation.

| | | | | | | |
|---|---|---|---|---|---|---|
| p0+p1 | p2 | 0 | 0 | 0 | 0 | 0 |
| 0 | p0 | p1 | p2 | 0 | 0 | 0 |
| 0 | 0 | p0 | p1 | p2 | 0 | 0 |
| 0 | 0 | 0 | p0 | p1 | p2 | 0 |
| 0 | 0 | 0 | 0 | p0 | p1 | p2 |
| 0 | 0 | 0 | 0 | 0 | p0 | p1+p2 |

Table 6.1: Approximate circulant matrix. The assumption holds for the central part of the matrix.

It must be noted that the circulant assumption is an *approximation* as the transition matrix generated by the Gaussian channel is **not** strictily circulant. The circulant assumption holds for most of the "central" symbols of the alphabet because the machine precision reduces the support of the probability mass function (the range of values where $p(x) > 0$) to a small range of about $4 \times \sigma$ and the channel is additive. Table 6.1 shows an example of how the approximation works.

The effects of such assumption have not been fully investigated, but currently this is the best available mechanism for making the DUDE-I work with Gaussian channels while running with practical computational requirements.[2]

### The parametric approach

Another approach to this problem is to make certain assumptions on the context-class-conditional input probability distributions $P_{X|\gamma}$. One possibility is to assume that $P_X$ is an instance of a given family of parametric distributions. As mentioned in Section 5.2, a good candidate is the two-sided geometric distribution (TSGD). Recall that the TSGD is defined as

$$P_{X|\gamma}(x) = \frac{1-\theta}{\theta^{1-s}+\theta^{s}}\theta^{-|x-\mu|}$$

where $\theta$ (decay term) and $\mu$ (center) are parameters of the distribution and $s = \lceil \mu \rceil - \mu$ is a term between 0 and 1. When $P_{X|\gamma}$ is modeled in this way, the parameters $\theta$ and $\mu$ can be obtained directly in terms of the mean and variance of $P_{Z|\gamma}$:

$$\begin{aligned} \mu_X &\overset{(a)}{=} \mu_Z = E_{P_{Z|\gamma}}[Z] \qquad (6.47) \\ \sigma_X &\overset{(b)}{=} \sigma_Z^2 - \sigma^2 = E_{P_{Z|\gamma}}[(Z-\mu)^2] - \sigma^2 \\ \theta &= \frac{\sigma_X + 1 - \sqrt{1+4\sigma_X}}{\sigma_X - 2} \end{aligned}$$

where $\mu_Z$ and $\sigma_Z$ are the mean and variance of $P_{Z|\gamma}$ and $\sigma$ is the *channel noise variance.* (a) and (b) are immediate using that the noise is white (additive, independent of the clean data and with zero mean). The full details of the derivation of 6.48 are given in Appendix C.

This alternative has the advantage of being much faster and to run in constant time when compared to the greedy algorithm.

[2]Disabling the circulant matrix assumption and doing a per-pixel inversion takes tens of hours to run on a 2.2GHz Pentium 4 HP Xeon station with 1GB of RAM for a $512 \times 512$ image. Using the assumption, the execution time on the same machine and for the same image reduces to about 5 minutes.

```
· for each conditioning class $\gamma$
    · Invert the channel in terms of the (noisy) prediction error distribution, $P_{E_Z|\gamma}$ using
      either the greedy algorithm or the parametric approach. Store the result as $P_{E_X|\gamma}$.
· for each noisy symbol $\mathbf{z}_i$
    · Take the precomputed input prediction error distribution for its conditioning class $\gamma_i$,
      $P_{E_X|\gamma_i}$.
    · Reconstruct the original context-class-conditional input distribution $P_{X|\gamma_i}$ using (5.4)
      for the current prediction $\hat{\mathbf{z}}_i$.
    · Apply the denoiser function in (4.4) to $P_{X|\gamma}$ for the current noisy pixel, $\alpha$. If $L_1$ or $L_2$
      norms are used, use instead (6.42) or (6.41) respectively.
```

Figure 6.18: Second pass of the DUDE-I for the Gaussian Channel.

Having only two parameters per distribution instead of $M$, the overall number of parameters to be estimated is greatly reduced and their estimations are more reliable. On the other hand, it puts heavy constrains on the shape of $P_{X|\gamma}$. This is related to the model cost problem described in Section 5, and some of its practical implications can be seen in Section 7.4.5.

#### Algorithm outline

The second pass of the DUDE-I for the gaussian channel is described in Figure 6.18

### 6.7.4 Denoising stage for the Salt & Pepper channel

Contrary to the Gaussian channel, the *Impulse* or so called *Salt & Pepper* channel is easily invertible and there is a very efficient closed form solution for it. It is easy to show that the inverse of (2.5) for a given parameter $\lambda$ is

$$\Pi^{-1} = \frac{1}{1-\lambda} \times \begin{bmatrix} 1-\frac{\lambda}{2} & 0 & \cdots & 0 & -\frac{\lambda}{2} \\ -\frac{\lambda}{2} & 1 & 0 & 0 & \vdots \\ \vdots & 0 & \ddots & 0 & -\frac{\lambda}{2} \\ -\frac{\lambda}{2} & 0 & 0 & 1 & -\frac{\lambda}{2} \\ -\frac{\lambda}{2} & 0 & \cdots & 0 & 1-\frac{\lambda}{2} \end{bmatrix}. \tag{6.48}$$

Given an output probability distribution for a given context $\gamma$, $P_{Z|\gamma}$, an Impulse Channel parameter $\lambda$, and an input/output alphabet $\mathcal{A} = \{0, \ldots, M-1\}$,

$$P_{X|\gamma} = \frac{1}{1-\lambda}[(P_{Z|\gamma}[0] - \frac{\lambda}{2}), P_{Z|\gamma}[1], \ldots, P_{Z|\gamma}[M-2], (P_{Z|\gamma}[M-1] - \frac{\lambda}{2})] \tag{6.49}$$

The above result has an intuitive interpretation: as $\lambda/2$ black (index 0) and $\lambda/2$ white (index $M-1$) counts are due to noise rather than to the clean sequence, to revert the channel effect means subtracting these amounts from the noisy distribution and re-normalizing (dividing by the resulting sum, namely $1-\lambda$). Furthermore, a closed expression for $P_{X|\gamma,\alpha}$ can also be obtained using (4.3). For $\alpha = 0$,

$$P_{X|\gamma,0} = \frac{[(1-\frac{\lambda}{2})(P_{Z|\gamma}[0] - \frac{\lambda}{2}), \frac{\lambda}{2}P_{Z|\gamma}[1], \ldots, \frac{\lambda}{2}P_{Z|\gamma}[M-2], \frac{\lambda}{2}(P_{Z|\gamma}[M-1] - \frac{\lambda}{2})]}{(1-\lambda)P_{Z|\gamma}(\alpha)} \tag{6.50}$$

for $\alpha = M-1$,

$$P_{X|\gamma,M-1} = \frac{[\frac{\lambda}{2}(P_{Z|\gamma}[0]-\frac{\lambda}{2}), \frac{\lambda}{2}P_{Z|\gamma}[1], \ldots, \frac{\lambda}{2}P_{Z|\gamma}[M-2], (1-\frac{\lambda}{2})(P_{Z|\gamma}[M-1]-\frac{\lambda}{2})]}{(1-\lambda)P_{Z|\gamma}(\alpha)} \tag{6.51}$$

and for the rest of the values,

$$P_{X|\gamma,\alpha} = [0, \ldots, 0, \overbrace{1}^{\alpha-th}, 0, \ldots, 0]. \tag{6.52}$$

Again, there is an intuitive interpretation of this result: as the only possible noisy symbols are 0 and $M-1$, any other observed symbol in the noisy sequence is *clean* and thus should be left *untouched* (i.e. $g(\alpha,\gamma)=\alpha$). In the case that the observed value corresponds to one of the noisy symbols, the resulting probabilities are essentially a scaled down version of (6.49) where the only position that has a relative change is that of the noisy value.

## Tail gathering

When prediction is used for Salt & Pepper channels, and because the noisy samples have a fixed, uncorrelated value, the prediction error statistics that correspond to the *black* and *white* pixels (the extremes) get *smeared.* To obtain an approximate picture of how this happens, consider the distribution of the prediction error $E$ for some conditioning class $\gamma$, $P_{E|\gamma}$ also conditioned on the three possible channel events: 'clean' when the sample gets out of the channel untouched, 'pepper' when it is substituted with $z=0$ and 'salt' when it is substituted with $z=M-1$,

$$\begin{aligned} P_{E|\gamma}(e) \quad = \quad & P(E=e|salt,\gamma)P(salt|\gamma)+ \\ & P(E=e|pepper,\gamma)P(pepper|\gamma)+ \\ & P(E=e|clean,\gamma)P(clean|\gamma) \end{aligned} \tag{6.53}$$

where $P(pepper|\gamma)$ and $P(salt|\gamma)$ are the probabilities of error of the channel as described in Section 2.2.2 for the conditioning class $\gamma$. Assuming that the context classification is not affected by the noise,[3] it can be further assumed that both probabilities are independent of the class $\gamma$ and thus equal to the global salt and pepper probabilities: $P(pepper|\gamma) = P(pepper) = \lambda/2$ and $P(salt|\gamma) = P(salt) = \lambda/2$. Thus (6.53) becomes

$$\begin{aligned} P_{E|\gamma}(E=e) \quad = \quad & \tfrac{\lambda}{2}P(E=e|salt,\gamma)+ \\ & \tfrac{\lambda}{2}P(E=e|pepper,\gamma)+ \\ & (1-\lambda)P(E=e|clean,\gamma) \end{aligned} \tag{6.54}$$

The prefiltered image will consist of clean samples and filtered samples. Carrying on with the assumption that the preclassifier accurately detects the noisy samples, the filtered samples will be based on other clean samples, and thus they will also be uncorrelated with the noisy samples they are substituting. As the prediction is built from this prefiltered image, it can be assumed that the predicted values will also be uncorrelated with the noisy values:

$$P(\hat{Z}=\hat{z}|salt,\gamma) = P(\hat{Z}=\hat{z}), \forall \hat{z} \in \mathcal{A}$$

[3]The goal of the prefiltering and preclassification blocks is to avoid this.

$$P(\hat{Z}=\hat{z}|pepper,\gamma)=P(\hat{Z}=\hat{z}),\forall\hat{z}\in\mathcal{A}$$

Using these results and the definition of the prediction error, $e=z-\hat{z}$,

$$\begin{aligned} P(E=e|pepper,\gamma) &= P(E=0-\hat{z}|\gamma)= && (6.55)\\ P(E=e|salt,\gamma) &= P(E=M-1-\hat{z}|\gamma) && (6.56)\\ & && (6.57)\end{aligned}$$

Assuming that the prediction is reasonably accurate, the distribution of the predicted values will be similar to the noisy distribution for the cases where the samples are uncorrupted

$$P(\hat{Z}=a|\gamma)\approx P(Z=a|\gamma,clean),\forall a\in\mathcal{A} \tag{6.58}$$

and thus

$$\begin{aligned} P(E=e|pepper,\gamma) &= P(E=0-\hat{z}|\gamma)\approx P(E=0-z|\gamma,clean) && (6.59)\\ P(E=e|salt,\gamma) &= P(E=M-1-\hat{z}|\gamma)\approx P(E=M-1-z|\gamma,clean) && (6.60)\\ & && (6.61)\end{aligned}$$

Finally, using (6.53) through (6.61),

$$\begin{aligned} P_{E|\gamma}(e) \approx\ & \tfrac{\lambda}{2}P(E=0-z|\gamma,clean)+\\ & \tfrac{\lambda}{2}P(E=M-1-z|\gamma,clean)+\\ & (1-\lambda)P(E=e|\gamma,clean)\end{aligned} \tag{6.62}$$

This approximation makes it possible to obtain a graphical representation of (6.53) by knowing the distribution of the noisy sequence $\mathbf{z}^{m\times n}$ and the distribution of $E$ for some $\gamma$. This approximate representation can be seen in Figure 6.19 and provides a justification for the *tail gathering heuristic* described below.

The final result is that, if no action is taken, equation (6.49) may yield *negative* probabilities in the *black* and *white* components. To avoid this, a heuristic scheme referred to as *tail gathering* was devised as a simple attempt to gather back the smeared statistics. This algorithm is described in Figure 6.20.

The use of this algorithm led to a consistent increase in the denoising performance in all the experiments performed with Salt and Pepper noise and is now considered an integral part of the second pass of the DUDE-I for Salt & Pepper noise.

The full second pass of the DUDE-I for the Salt & Pepper channel is developed in Figure 6.21.

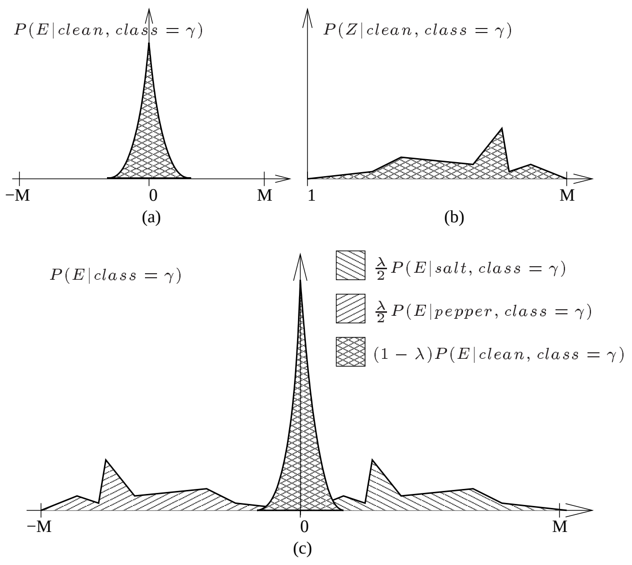


Figure 6.19: Approximate shape of the conditional prediction error distribution $P_{E|\gamma}$ when the noisy image $\mathbf{z}^{m\times n}$ is the output of an Impulse Channel of parameter $\lambda$.

```
· set $d = \frac{\lambda}{2} - P_{Z|\gamma}[0]$
· set $i = 1$
· while $d > 0$ and $i < M - 1$
    · if $P_{Z|\gamma}[i] < d$,
      $P_{Z|\gamma}[0] = P_{Z|\gamma}[0] + P_{Z|\gamma}[i]$;
      $d = d - P_{Z|\gamma}[i]$;
      $P_{Z|\gamma}[i] = 0$;
      $i = i + 1$
    · else
      $P_{Z|\gamma}[0] = P_{Z|\gamma}[0] + d$;
      $P_{Z|\gamma}[i] = P_{Z|\gamma}[i] - d$;
      END.
· set $d = \frac{\lambda}{2} - P_{Z|\gamma}[M-1]$
· set $i = M - 2$
· while $d > 0$ and $i > 0$
    · if $P_{Z|\gamma}[i] < d$
      $P_{Z|\gamma}[M-1] = P_{Z|\gamma}[M-1] + P_{Z|\gamma}[i]$;
      $d = d - P_{Z|\gamma}[i]$;
      $P_{Z|\gamma}[i] = 0$;
      $i = i - 1$
    · else
      $P_{Z|\gamma}[M-1] = P_{Z|\gamma}[M-1] + d$;
      $P_{Z|\gamma}[i] = P_{Z|\gamma}[i] - d$;
      END.
```

Figure 6.20: Tail Gathering algorithm.

```
For each $\mathbf{z}_i \in \mathbf{z}^{m\times n}$
    · Take the conditioning state for current pixel $\gamma_i$ from the conditioning map $\gamma^{m\times n}$.
    · Retrieve the prediction error distribution for it, $P_{E|\gamma_i}$.
    · Obtain $P_{Z|\gamma_i}$ from $P_{E|\gamma_i}$ using (5.4)
    · Obtain the estimated channel input distribution $P_{X|\gamma_i}$ using (6.49).
    · Add the conditioning on the current noisy sample, $\mathbf{z}_i$, according to one of (6.50), (6.51) or (6.52).
    · Compute $\mathbf{x}_i^*$ as symbol which yields minimum expected loss for $\gamma_i$ and $\alpha = \mathbf{z}_i$ using (4.4) or one of its faster forms (6.41) or (6.42) if $L_2$ or $L_1$ norms are used as the cost function.
```

Figure 6.21: Second pass for the Impulse Channel.

Since inverting this channel is relatively inexpensive in terms of computation when compared to the rest of the denoising process, it can be done for each single pixel with no noticeable increase in computation time. This eliminates the need for approximations such as the circulant matrix assumption needed for Gaussian Channels or the precomputation of the denoising function for the possible different combinations of noisy sample value, prediction and context class, $(\mathbf{z}_i, \hat{\mathbf{z}}_i, \gamma_i)$, that may appear in the image.

## 6.7.5 Denoiser function for the $q$-ary channel

As for the impulse channel and its variants, the inverse of the q-ary channel can be computed directly and expressed in terms of the channel parameters as well:

$$\Pi^{-1} = \frac{1}{1-\lambda} \times \begin{bmatrix} c & d & \dots & d \\ d & c & \dots & d \\ \vdots & \vdots & \ddots & \vdots \\ d & d & c & d \\ d & d & \dots & c \end{bmatrix}. \tag{6.63}$$

where $c = \frac{M+p-2}{Mp-1}$ and $d = \frac{p-1}{Mp-1}$ with $p = 1 - p_{err}$ and $M = |\mathcal{A}|$. This yields a simple closed form for the calculation of $P_{X|\gamma}$ and $P_{X|\gamma,Z=z}$.

$$P_{X|\gamma}[i] = (c-d)P_{X|\gamma}[a] + D, \forall i \in \mathcal{A} \tag{6.64}$$

$$P_{X|\gamma,Z=z}[i] = \begin{cases} AP_{X|\gamma,Z=z}[i] & , \quad i = z \\ BP_{X|\gamma,Z=z}[i] & , \quad i \neq z \end{cases} \tag{6.65}$$

where $A = 1 - p_{err}$ and $B = p_{err}/M$.

While testing this channel, the initial denoised images showed very noticeable noisy pixels that where left untouched by the algorithm, while most of the less noticeable ones were correctly denoised.

After further investigation, it was observed that the tail gathering procedure described in (6.44) was the source of the problem.

To explain this fact, first observe that the coefficients of (6.65) are of very different orders of magnitude. For instance, for an 8-bit alphabet and $p_{err} = 10\%$, $A = 0.9$ and $B = 0.1/256 = 0.0004$. Thus, even if the prediction distribution is highly concentrated around 0, a small tail can grow to a point where it dominates the solution. This only happens when $z = 0$ or $z = M-1$ as the tails are gathered at those values. Figure 6.22 shows this concept for a real case

For the $q$-ary symmetric channel, the **tail gathering** algorithm, which had improved the results for the other non-additive noise types, produced the undesired effect of amplifying the influence of the outliers when their values was exactly in the borders of the alphabet.

As a result, when using this channel, the best solution was to disable the tail gathering algorithm.

### 6.7.6 Denoising function cache

The original DUDE implementation proposes the precomputation of the denoiser function for each possible combination of its arguments which are the noisy context of the noisy sample and the value of the noisy sample itself. When working with continuous tone images the size of the alphabet represents a problem for this approach, namely:

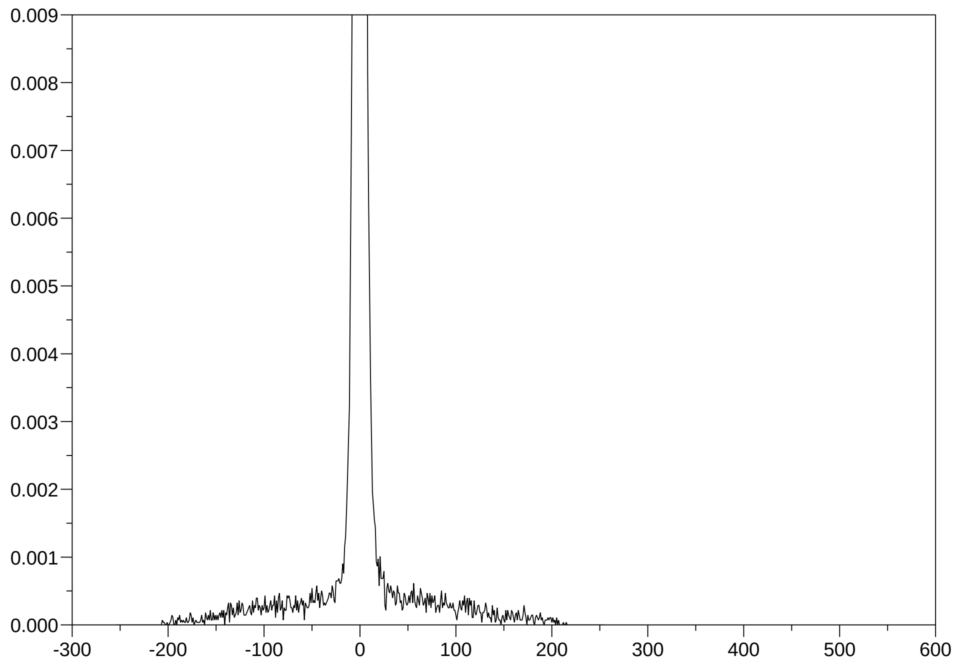
1. The number of possible contexts is $|\mathcal{A}|^K$ for a context of size $K$.
2. The number of possible noisy symbols, $z$, is $|\mathcal{A}|$.

This results in $|\mathcal{A}|^{k+1}$ possible combinations.

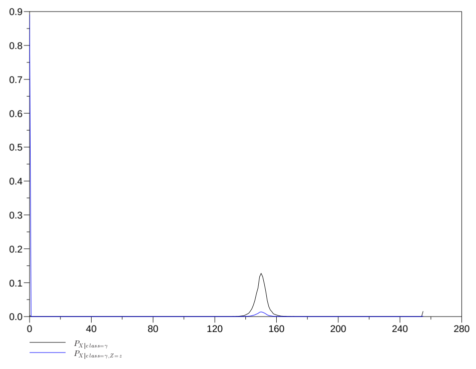
The first of the two problems is already reduced by the context classification approach used in this work, where the possible context classes $\gamma \in \Gamma$ and thus the number of possible combinations is reduced to $|\Gamma||\mathcal{A}|$, provided that $|\Gamma| << |\mathcal{A}|^k$. However, the use of prediction makes the denoising function depend on yet another variable: the predicted noisy symbol $\hat{z}$. Because of this, the final size of the cache would be $|\Gamma||\mathcal{A}|^2$ which, for the common 8-bit grayscale images, and $|\Gamma| = |\mathcal{A}| = 256$ would be $256^3 = 16777216$.

One way to reduce this problem is by observing that, if the prediction errors are highly concentrated around 0, a partial cache which includes only those combinations of $(z, \hat{z}, \gamma)$ for which $|z - \hat{z}| < \epsilon$ can still cover the majority of the cases while reducing its size to $|\Gamma||\mathcal{A}|(2\epsilon + 1)$ if $2\epsilon + 1 << \mathcal{A}$.

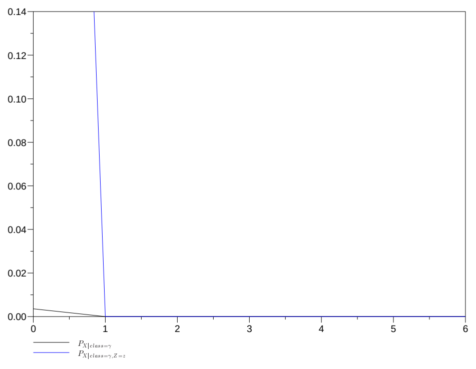
The inclusion of this strategy in the augmented DUDE yields an important reduction in computational cost and at the same time reduces the memory needed to a degree where it is not significant with respect to the requirements of the other components of the algorithm.



(a) Prediction error distribution.



(b)



(c) Zoomed version of (b)

Figure 6.22: Sample distributions for the pathological case of $z = 0$. Note how the tail of $P_{X|class=\gamma}$, which is much smaller at that point, gets amplified when the conditioning on $z$ is added to yield $P_{X|class=\gamma,Z=z}$.

# 7 Results and discussion

## 7.1 Design of the experiments

This chapter presents the results of applying the proposed solution to the different noise models described. When available, the current state of the art resutls for each noise model are presented, and the discussion continues by comparing them to the ones obtained with the DUDE-I for each modeling approach (Legacy, Napkin or Combined LBG+Napkin).

All the results, excluding those of Section 7.5, are based on simulated noise over the images, so there is no noise or type parameter estimation implied, and thus the noise model parameters assumed by the DUDE-I are the "real" ones. Because of this, a *sensitivity analysis* of the DUDE-I when tuned to the wrong channel parameters is also presented at the end of the discussion of each type of noise.

The results are presented in terms of PSNR (Peak Signal to Noise Ratio) with respect to the clean image, as it is the standard objective measure of denoising performance used in the literature. Recall from Section 2.3 that given a clean image $\mathbf{x}^{m\times n}$ and a noisy version of it $\mathbf{z}^{m\times n}$, the PSNR is defined as

$$\mathrm{MSE}(\mathbf{z}^{m\times n}) = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}_i - \mathbf{z}_i)^2.$$

The PSNR is expressed in dB (deciBells). The number of significant digits in all the results is 1, as the experiments indicate that differentrandom simmulations (different random seeds) yield a variance of 0.1dB in all the results. This is discussed later in Section 7.2.6 and Section 7.4.7.

The PSNR is the standard measure for comparing the performance between different algorithms. However, it is desirable to have a measure of performance which does not require the knowledge of the clean image, since the latter may not be available (such as in a real problem where the noise is not simmulated), and thus the selection of the parameters which give the best denoising performance could not be based on the PSNR. Motivated by the results in [34, Sec. VII-B], the *compressibility* arises as a possible measure which has the desired properties:

- As a measure the compressibility of an image we use the average bpp (bits per pixel) obtained by compressing it using the lossless compression algorithm JPEG-LS. Clearly, this measure does not depend on the knowledge of the clean image.
- The fact that the compressibility is a good measure of denoising performance is shown in [34, Sec. VII-B] for the binary DUDE in halftone images and for the size of the context. In

the latter work, an empirical experiment shows that the local minimum of the compressibility roughly coincides with the local minimum of the difference between the denoised image and the clean image. As the results in this chapter will show, this empirical result extends to the other parameters present in the DUDE-I.

When comparing performances, the best value for a given setting is shown in **bold face**.

The test images are taken from two standard test suites: the one mantained by the Signal & Image Processing Institute of the University of Southern California[1] which contains most of the classical images used in image processing papers, and the one used in the development of the JPEG-LS standard[2].

Results will be presented in tabular and graphic form. For the sake of brevity, the tabulated results are given for a small representative subset of the full test suites: the smaller *Lena*, *Barb*, *Boats* and *Bridge*, of about $1/4MP$ (MP stands for Mega Pixel,i.e., 1 million of pixels) are shown in Figure 7.1, and the bigger *Bike* with $4MP$ is shown in Figure 7.2.[3]

For each type of noise, the best results are shown for a small set of typical parameters used in the literature. Then, the algorithm parameters which yield the best results are specified, along with the tests that were performed to obtain them. The latter set of tests are shown only for one noise parameter, except for a few specific cases.

When denoised images are shown for visual inspection, they are accompained by the noisy and clean images, and also for the image of the *absolute difference* between the denoised and the clean image. Darker values in this image indicate higher differences, and clearer regions indicate good denoising performance. This serves as an additional tool to study the performance of the algorithms when visual inspection of the denoised image alone is not enough.

The different types of noise can be divided into two groups: non-additive noise (even and uneven Salt and Pepper, Z-Channel, $q$-ary Symmetric) and additive noise (Gaussian). Some tests were performed only once per group, using the Salt and Pepper channel as the representative of the latter group, and (obvioulsy) the Gaussian channel for the former.

[1]http://sipi.usc.edu/services/database/

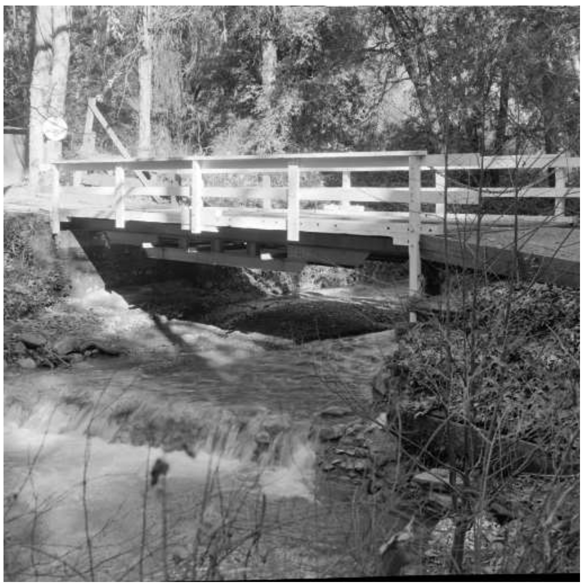
[2]http://www.jpeg.org/

[3]All the experiments presented in this chapter were performed on a larger subset of the SIPI suite, and for the Napkin modeling, also for the JPEG-LS set. The selection of the best parameters was based on the full results.

(a) Boats (720 × 576). This is an interesting image which is rich in edges at different angles.

(b) Lena (512 × 512). This widely used image is notorious for its smoothness, which makes it an "easy" image for denoising purposes.

(c) Barbara (720 × 576). This imagewas designed to include small details and dominated by fine textures. It is thus a more challenging image.



(d) Bridge (512 × 512). The version included in the SIPI suite of this image has been subject to contrast enhancement using histogram equalization [9, pp. 146—152], thus including many pure-black and pure-white pixels.

Figure 7.1: Test images.

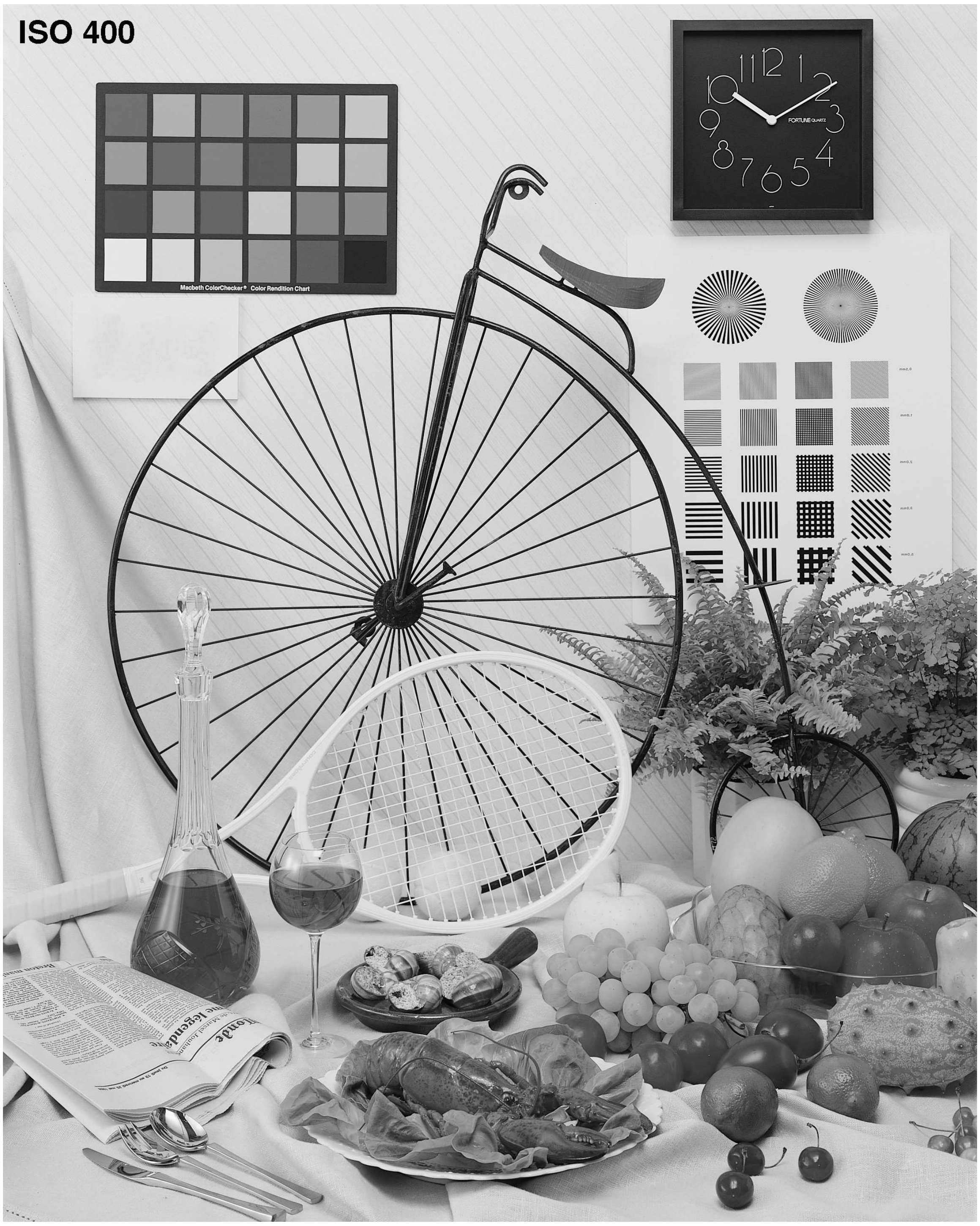


Figure 7.2: "Bike" (2048 × 2460).This large image belongs to the JPEG-LS test suite and was designed to include many different patterns.

| $\lambda$ | image | noisy | SM | GIO | CSAM | MND-DP |
|---|---|---|---|---|---|---|
| 10% | lena | 15.4 | **39.9** | - | 39.2 | - |
| | boats | 15.4 | **38.5** | - | - | - |
| | bridge | 15.2 | 33.2 | - | **37.2** | - |
| | barb | 15.3 | **33.5** | - | - | - |
| 30% | lena | 10.7 | 33.9 | **35.7** | 34.3 | - |
| | boats | 10.6 | 32.1 | **34.6** | - | - |
| | bridge | 10.5 | 27.9 | - | **31.5** | - |
| | barb | 10.6 | **28.0** | - | - | - |
| 70% | lena | 7.0 | 16.7 | - | - | **29.3** |
| | boats | 7.0 | **16.7** | - | - | - |
| | bridge | 6.8 | 15.8 | - | - | **25.0** |
| | barb | 6.9 | **16.0** | - | - | - |

Table 7.1: Reference base for impulse noise removal. The noisy image PSNR is also included for further comparison.

## 7.2 Salt and Pepper noise

### 7.2.1 Reference results

The main results are presented for the cases $\lambda = \%10$, $\lambda = \%30$ and the more extreme case of $\lambda = \%70$, which are common settings found in the literature. For the sake of brevity, only the best results will be shown for all the three parameters. For the rest of the experiments, the "average" case $\lambda = 30\%$ will be used.

As a basis for the discussion, the results of applying a simple *selective median filter* (SM) to the test images and the results from other works in the field are summarized in Table 7.1. The keys to the column labels are:

**GIO** Previous version of the DUDE for continuous tone images [18]. This is basically the Legacy Scheme described in Section 6.5 without enhancements such as the prediction error distribution clipping described in Equation Section 6.43, the Tail Gathering algorithm described in Section 6.7.4, or the recursive prefiltering scheme (Section 6.3.2).

**CSAM** Median filtering of noise using the co-occurence matrix method for impulse noise detection [24].

**MND-DP** Median Noise Detection with Detail Preserving [2].

The best values obtained in each case will be used as the reference against which we will compare the proposed algorithms under the same conditions.

### 7.2.2 Legacy results

Table 7.2 shows the best results obtained by using the Legacy modeling scheme using an Average Filter to predict the center sample, when applied to images corrupted by 30% impulse noise. The best configuration in this case was found to be the following:

| image | reference | Legacy |
|---|---|---|
| lena | 35.7 | **37.5** |
| boats | 34.6 | **36.7** |
| barb | 28.5 | **33.9** |
| bridge | **31.5** | 30.0 |

Table 7.2: Best results in terms of PSNR for the Legacy modeling scheme under $\lambda = 30\%$.

| pred. | barb | boats | bridge | lena |
|---|---|---|---|---|
| average | 4.9 | 4.2 | 5.6 | 4.3 |
| median | 4.9 | 4.2 | 5.6 | 4.3 |

| pred. | barb | boats | bridge | lena |
|---|---|---|---|---|
| average | 32.1 | 35.5 | 29.6 | 36.7 |
| median | 31.7 | 35.0 | 29.4 | 36.4 |

Table 7.3: Performance of Legacy vs. prediction filter. Left: Compressibility (average bits per pixel); Right: Denoising performance (PSNR in dB)

- The trivial preclassification scheme is used to obtain a noise mask.
- A 256 cluster set is obtained after 25 LBG iterations on raw $5 \times 5$ pixel contexts present in the prefiltered sequence.
- The filter used for prediction is an Average Filter applied to a $5 \times 5$ square context.
- The cost function is $L_2$.
- Four recursive applications of the DUDE-I as a prefilter, with the initial prefilter set to a Selective Median filter over square windows of $5 \times 5$ pixels.

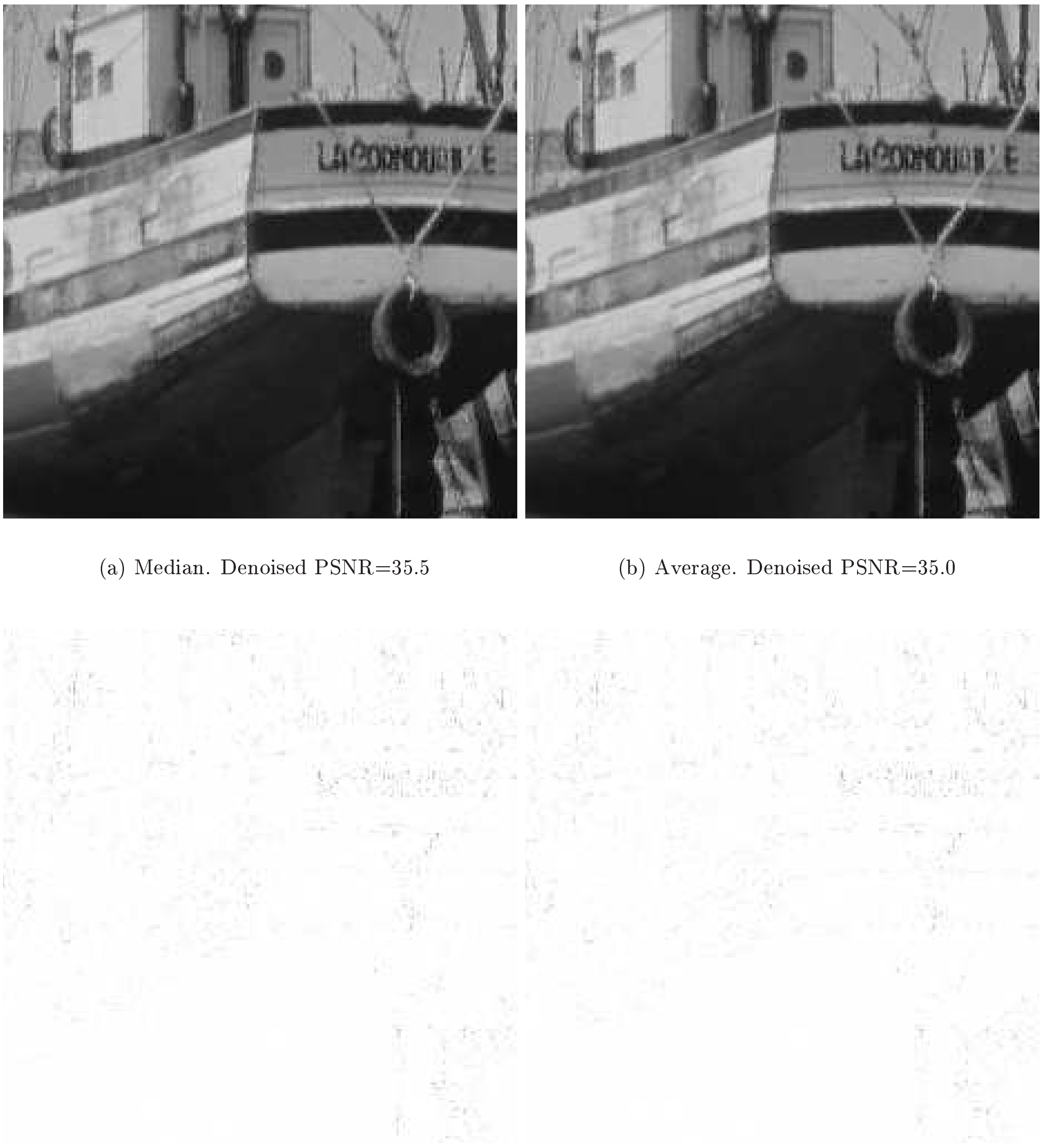
### Selection of the parameters

These tests show how the performance varies with respect to the different parameters of the Legacy Scheme.[4] The following parameters are of special interest:

**Predictor** The Legacy Scheme uses a simple sliding neighborhood filter to predict each pixel. Figure 7.3 shows the results of denoising using a window average and a windom median filter as a predictor. The special case of the Combined Modeling will be described in detail in Section 7.2.4.

As can be seen, despite the $0.5dB$ difference, there is no noticeable visual difference between the two images. Only a detailed inspection of the whole image (impossible to observe here) reveals that the difference lies in the borders of the image. This is also dependent on the way in which the image samples are extrapolated for the context samples that fall out of the image (which happens in the borders). As a simple replication of the border pixels is used as the strategy for the window samples falling outside the range of the image, the median filter could be more affected than the average. Note also that the prediction is performed over an already median-prefiltered image. As this advantage of the Average over the Median prediction is confirmed in all the test images, the Window Average was chosen.

[4]Note that, in each experiment, the results are obtained with no recursive prefiltering performed. Obviously, this does not apply for the experiments whose subject is the recursive prefiltering behavior.



(a) Median. Denoised PSNR=35.5

(b) Average. Denoised PSNR=35.0

(c) Absolute difference of Median.

(d) Absolute difference of Average.

Figure 7.3: Legacy performance vs. predictor.

**Number of context clusters** In the augmented framework, this number determines the number of free parameters in the probabilistic model used for the image to be denoisedm and is related to the model cost described in Section 5.1. In the baseline DUDE, the number of paramerters depends on the number of possible contexts, which in turn is determined by the size of the context template used. As described in Section 4, this results in a restriction on the size of the template if asymptotic optimality is to be guaranteed. Thus, it is natural to believe that the optimal number of context clusters should be related to the size of the images to be denoised (not for a particular image but for a given size). The results shown in Figure 7.4 seem to confirm the existence of an optimum value for the number of context clusters, as all of the images are of similar size. Note that the compressibility heuristic, which is described in [34, Sec. VII-B], gives an optimum which coincides with the optimum obtained by computing the PSNR with the clean image also in this case. This parameter also has an important impact in the computational complexity, requiring $O(m \times n)$ additional operations for each class defined. So, it is desirable to keep it at a minimum. Based on this observation and the results in Figure 7.4, a value of 256 was chosen as a good tradeoff.

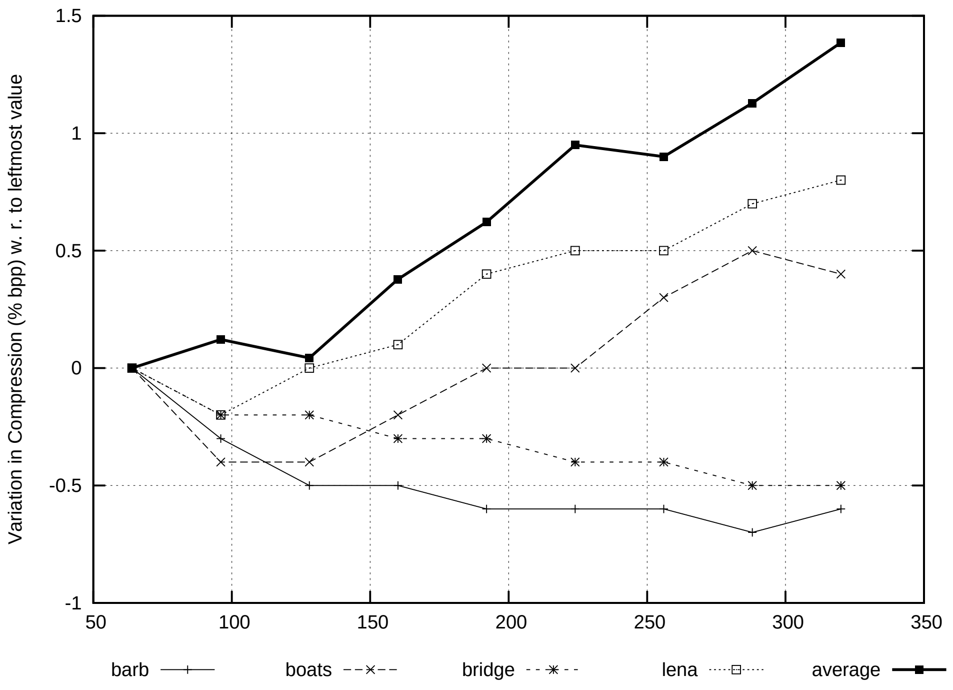
**Size of the contexts** As the number of context classes is not affected by this parameter, it does not play a role in the model cost as it does in the baseline DUDE algorithm. However, it affects the characterization of the contexts, and affects linearly the complexity of the algorithm. From the results in Figure 7.5 it can be seen that a context size of $3 \times 3$ pixels yields the best denoising performance in all the cases. However, as the Legacy Scheme is currently limited in practical terms (not theoretically) to a small range of image sizes, it cannot be said that this size of context is optimum for other sizes of images.

**LBG iterations** This parameter implies a tradeoff between the computational complexity (number of operations) and the representativeness of the cluster centers. More iterations allow the LBG algorithm to better approach a stable solution. As this stage of the algorithm dominates the total execution time of the Legacy Scheme, with each iteration taking $O(m \times n)$ operations, the cost of each new iteration is high and it is desirable to keep the number of iterations at a minimum. Figure 7.6 shows the effect in terms of PSNR for the test images and Figure 7.7 shows a detail of Boats. Based on the results a number of 25 iterations was chosen as a good tradeoff between denoising performance and execution time.

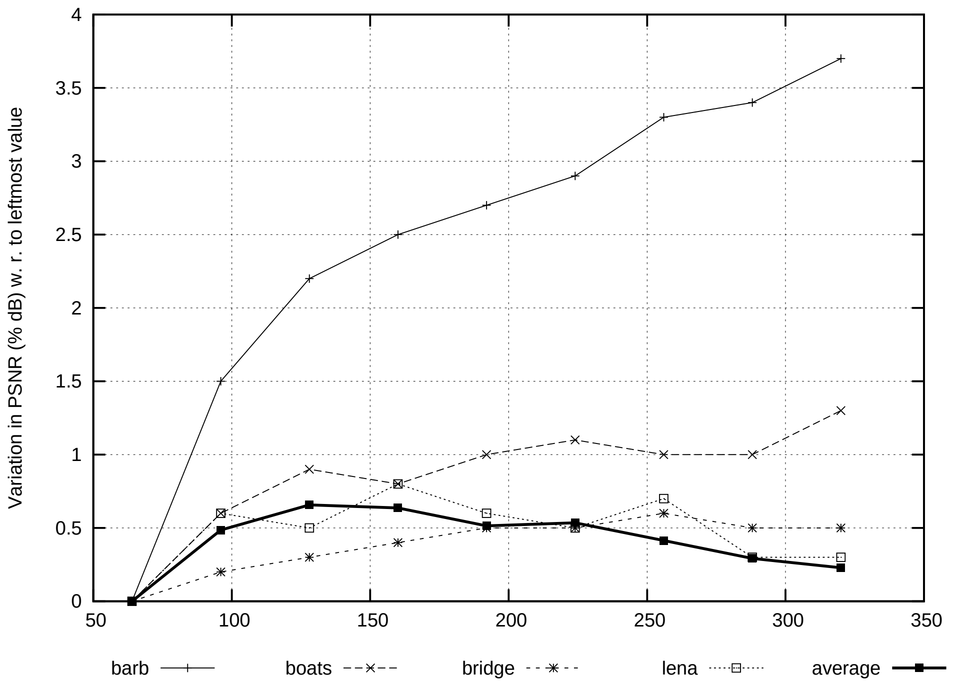
**Canonical mapping** This procedure was presented in Section 6.5 as a way to join similar contexts found at different orientations across the image. Figure 7.8 shows the effect of its application in the way in which it affects the context classification of the pixels of a sample image. Although only shown for Lena, this mapping has improved the performance for all the images of the test suite and thus it is activated by default.

**Number of prefiltering recursions** The recursive prefiltering can improve the performance significantly, at the cost of multiplying the computation time by the number of recursions. The results, however, do not increase in a monotonic way but reach a saturation point which varies with the image and, as will be seen later, the modeling scheme (context and prediction). Figure 7.9 shows the results for a maximum of five iterations.

**Summary of the Legacy Scheme for Salt and Pepper** The preceding results in this section show a significant improvement in the results of the Legacy Scheme when compared to the results obtained in [18]. For the majority of the images and channel parameters studied,

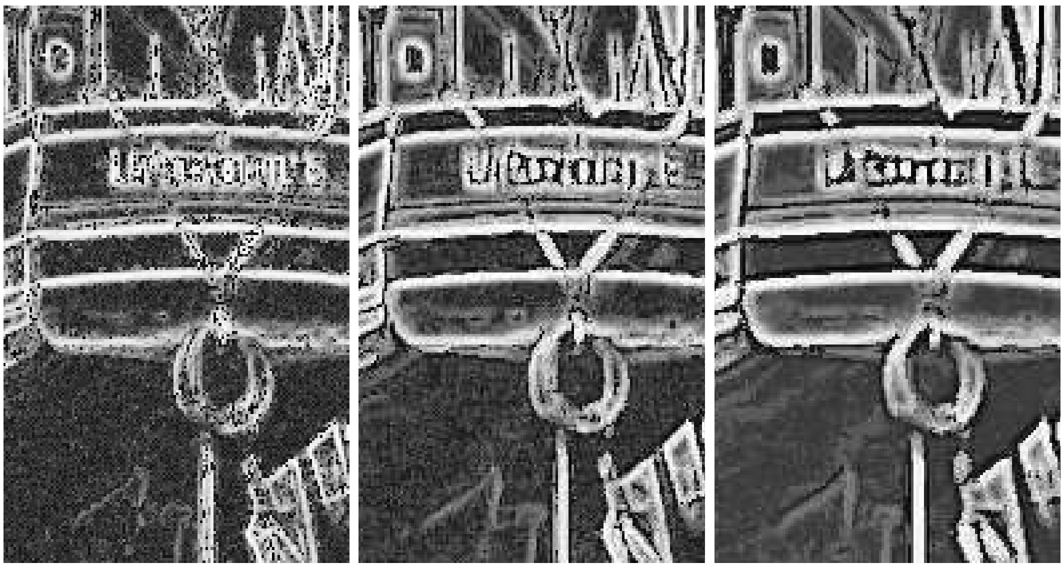


(a) Compressibility.



(b) Fidelity.

Figure 7.4: Legacy performance vs. number of clusters.



(a) Contexts of $3 \times 3$ pixels. Denoised PSNR=33.8

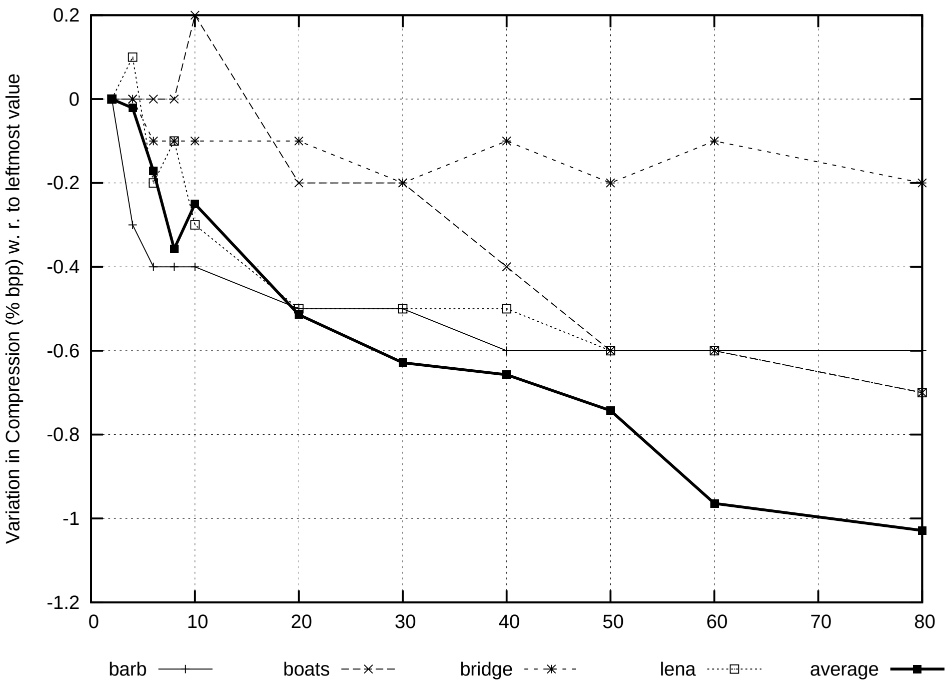
(b) Contexts of $5 \times 5$ pixels. Denoised PSNR=36.4

(c) Contexts of $7 \times 7$ pixels. Denoised PSNR=35.4

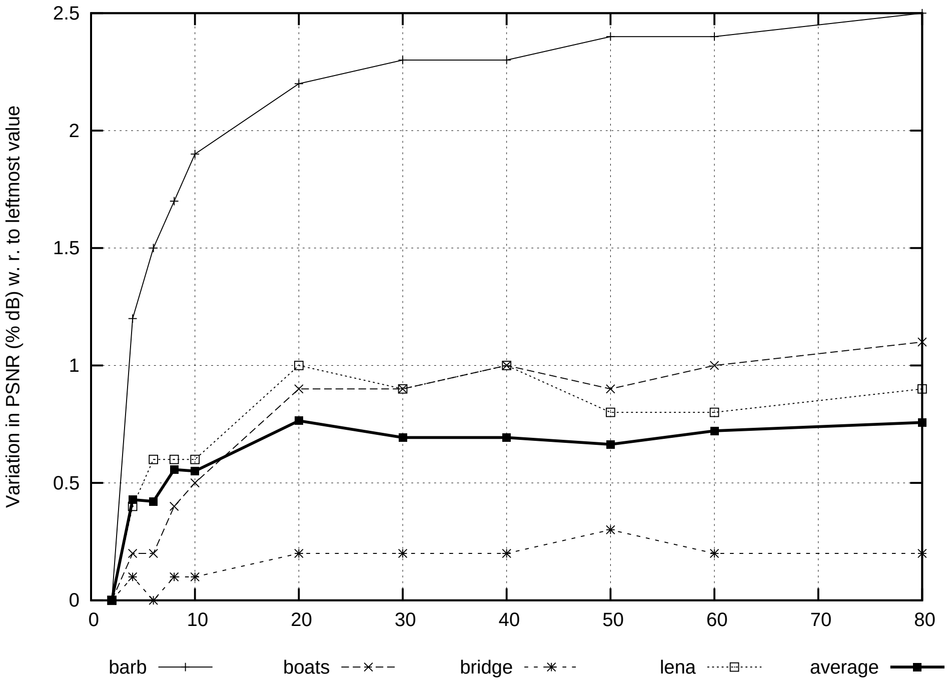
Figure 7.5: Legacy performance vs. size of the contexts.

the results also surpass the best available results in the literature and, in many cases, by ample margins (over $2dB$ of PSNR).

On the counter side, this scheme has high computational resources requirements for the current standards which make it impractical for images of over 1MP.

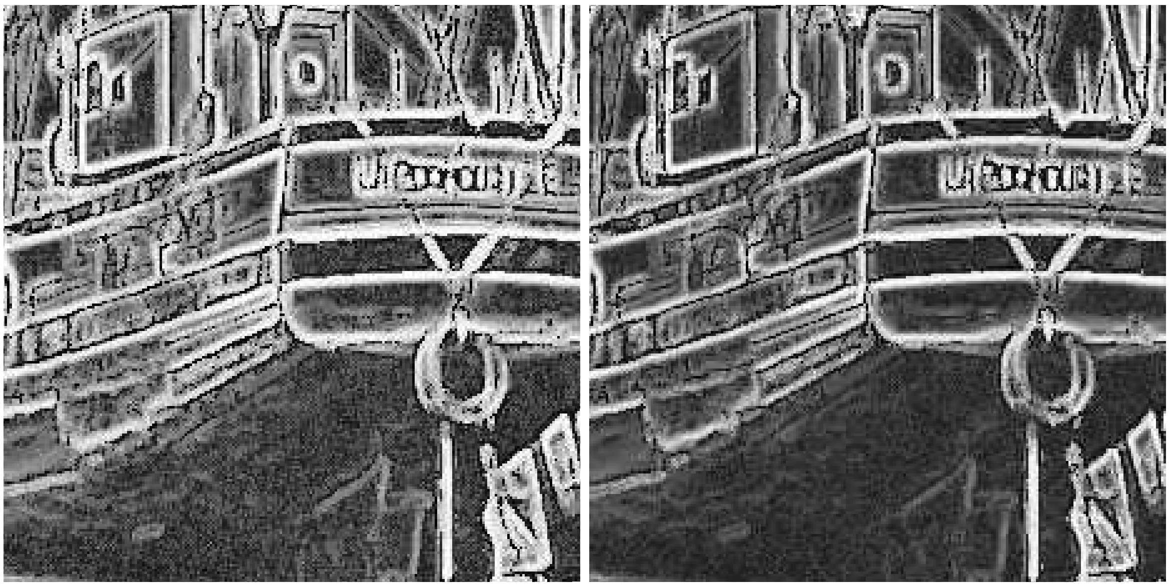


(a) Compressibility.



(b) Fidelity.

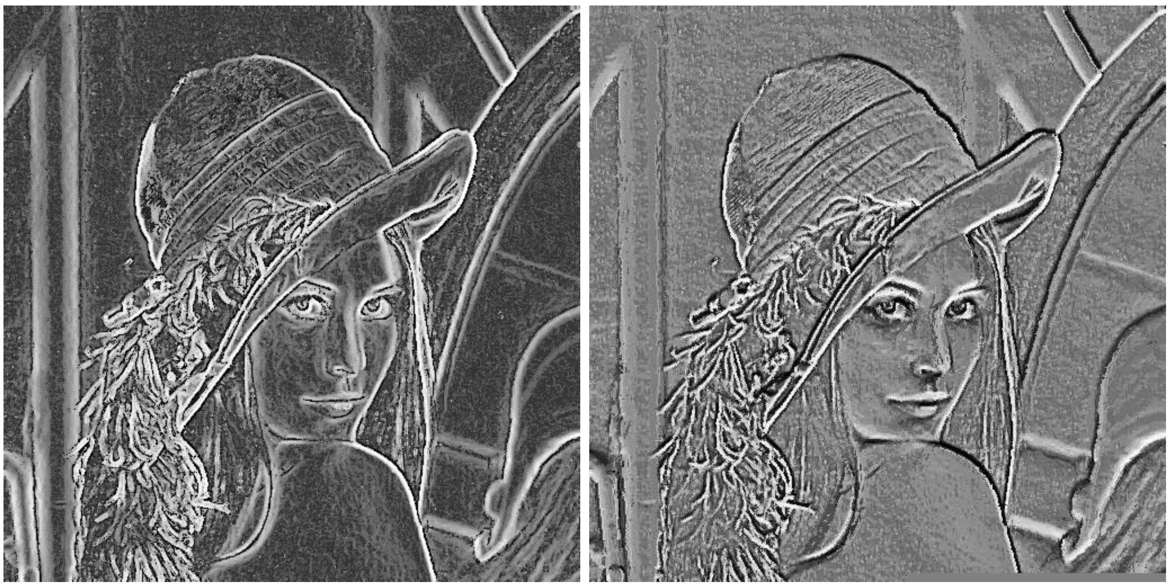
Figure 7.6: Legacy performance vs. LBG iterations. (a) shows the percentual variation in compressibility with respect to the smallest (leftmost) value and (b) shows the percentual variation in denoised PSNR w.r.t. the leftmost value



(a) Conditioning map after 8 iterations. Denoised PSNR=36.2

(b) Conditioning map after 50 iterations. Denoised PSNR=**36.5**

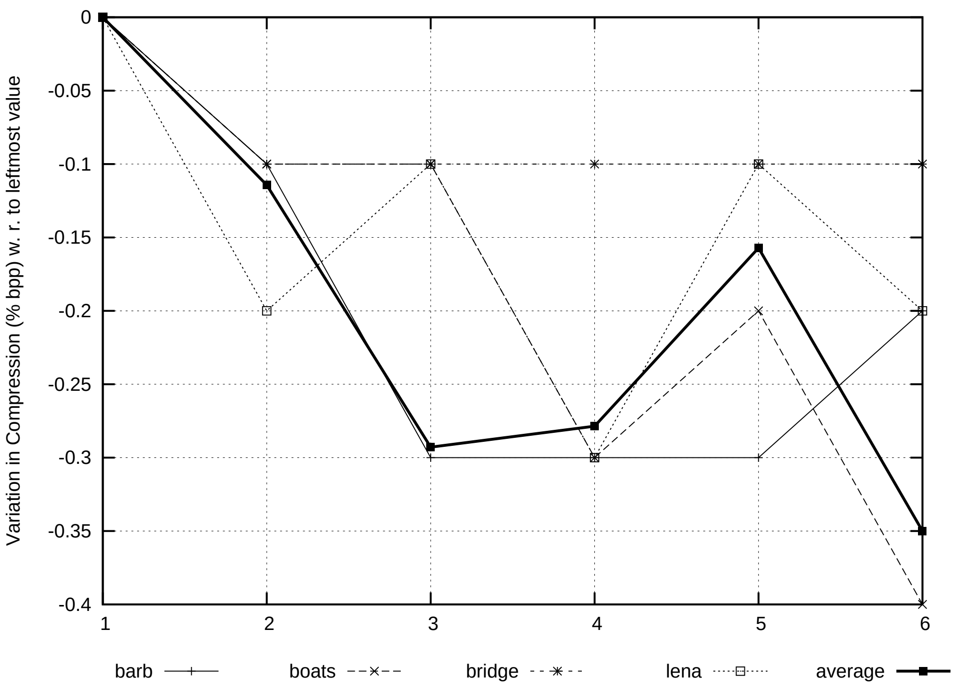
Figure 7.7: Legacy vs. LBG iterations, detail of Boats. (a) and (b) show how the conditioning map looks after 8 and 50 iterations respectively. Notice how the map is more "ordered" as the iterations are increased.



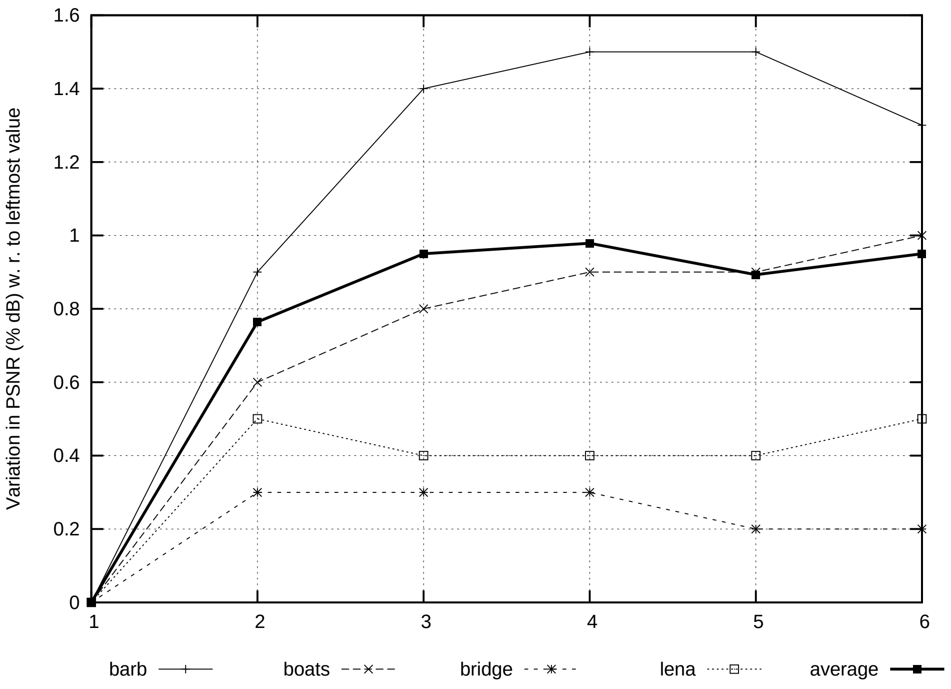
(a) With canonical mapping. **PSNR=36.7**

(b) Without canonical mapping. PSNR=36.2

Figure 7.8: Conditioning maps for Lena when modeling using (a) and not using (b) canonical mapping.



(a) Compressibility.



(b) Fidelity.

Figure 7.9: Legacy performance vs. recursive prefiltering applications.

### 7.2.3 Napkin

The best results obtained for the Napkin Modeler described in Section 6.6, for the case $\lambda = 30$, are shown in Table 7.4, and three sample denoised images presented in Figures 7.10, 7.11, and Figures 7.12 through 7.14. These were obtained with the following configuration:

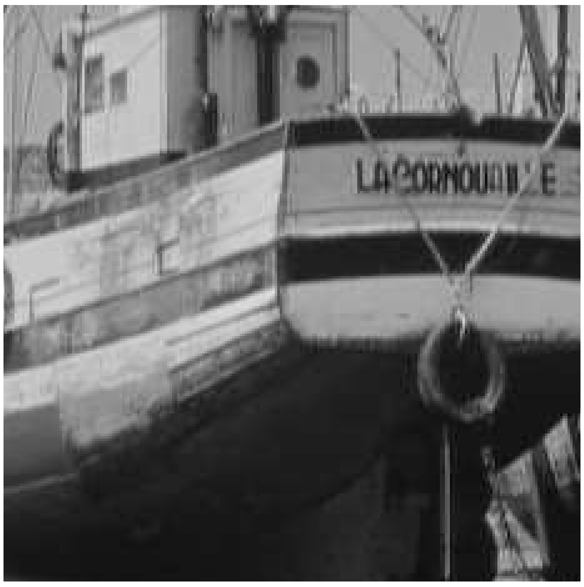
- Th Activity level (AL), defined in Section 6.6.3, is quantized into 8 levels (i.e., to 3 bits) using the quantization algorithm defined in Section 6.6.3 and the resulting value is used to define 8 possible probability conditioning states.
- The same 3 AL bits are combined with 8 texture bits (TB) of the Texture Bitmap defined in Section 6.6.3 to produce 2048 context-dependent bias cancellation terms. As define, the resulting prediction classes are refinements of the 8 probability conditioning classes.
- The Average Napkin prediction variant (Section 6.6.5) is used with a gradient threshold of 8% of the maximum possible gradient magnitude to determine the flatness of each wing.
- Seven recursive applications of the DUDE-I as a prefilter are used, with the initial prefilter set to a selective median over $5 \times 5$ windows.

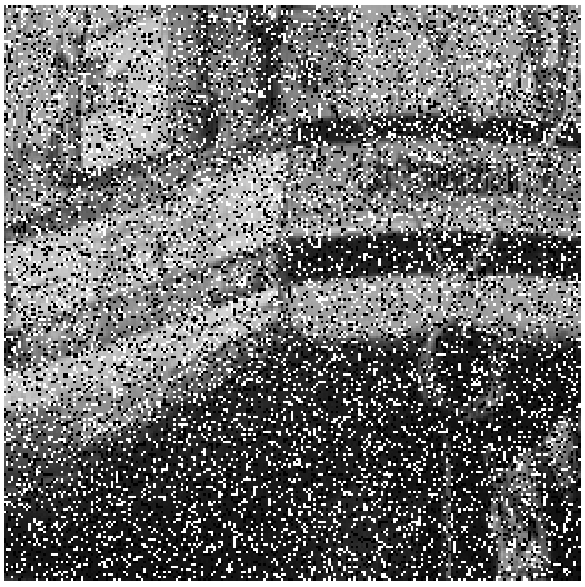
| image | reference | Legacy (x2) | Napkin (x7) |
|---|---:|---:|---:|
| lena | 34.3 | 37.7 | **38.2** |
| boats | 32.2 | 36.8 | **38.3** |
| bridge | **31.5** | 30.0 | 30.6 |
| barb | 28.0 | **33.7** | 32.2 |
| bike | 26.0 | - | **29.6** |

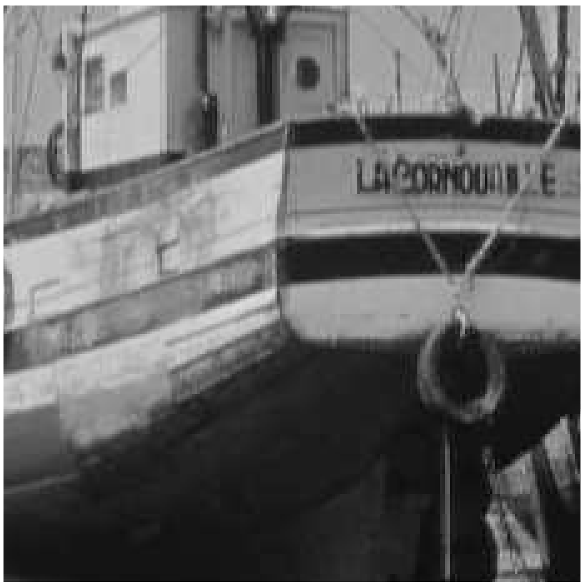
Table 7.4: Best Napkin results for $\lambda = 30$.

The results for the Napkin modeling scheme are significantly better than those of the Legacy scheme for all the images in the test suite (also those not shown here) with the exceptions of “Barb” and “Barb2”, where the Legacy scheme is clearly better. This gives an overall advantage to the Napkin modeling scheme but also signals a potential pitfall of the modeling algorithm when confronted with high frequency patterns such as those present in Barb.

As a way to isolate the problem found with the two versions of “Barb”, a third modeling scheme was defined which combined the context classification method of the Legacy Scheme (LBG) with the Napkin predictor. This scheme was called *Combined LBG/Napkin.* The results for this scheme (which ended up being the best in terms of denoising performance) are given later in Section 7.2.4.

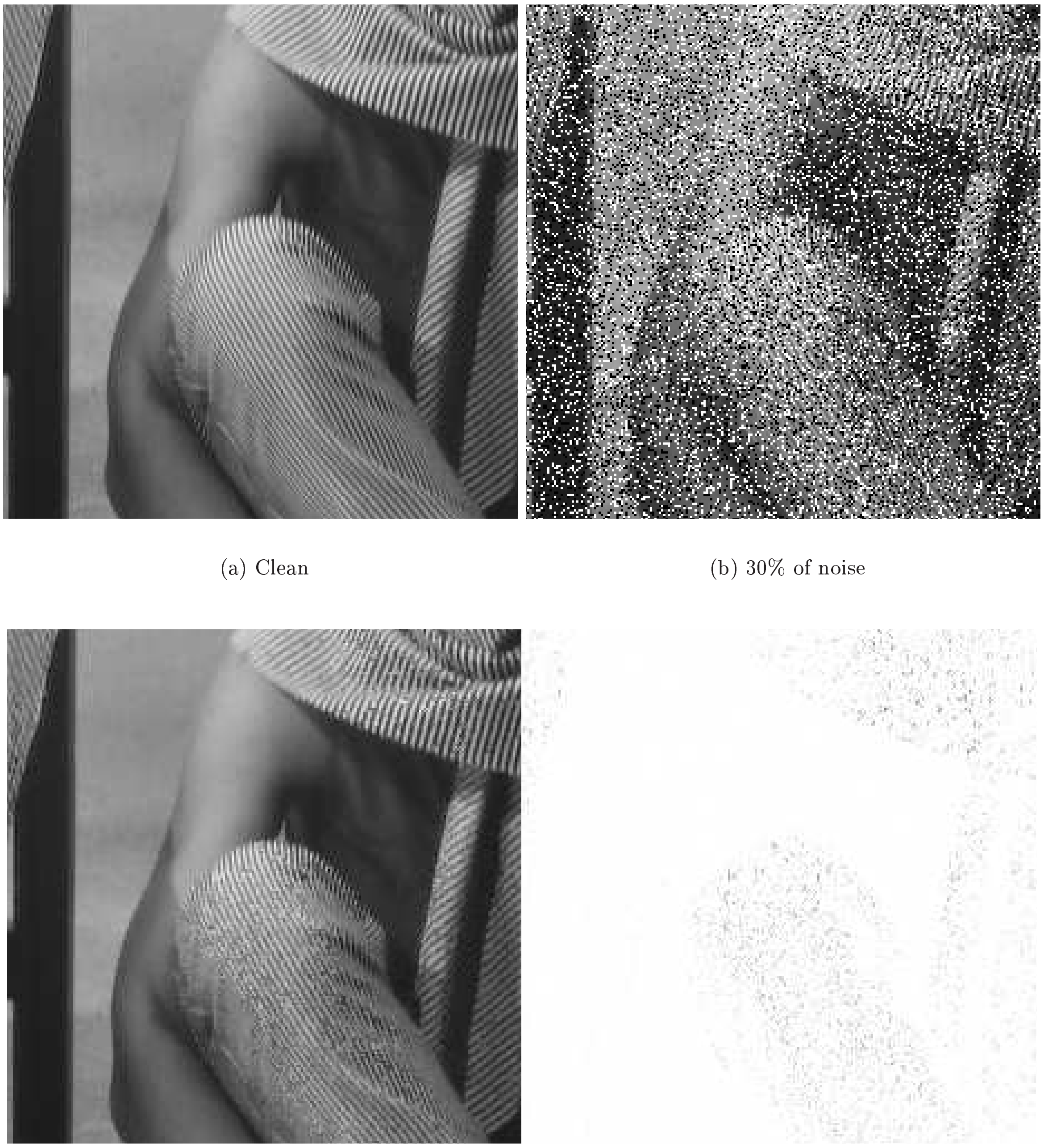

(a) clean


(b) 30% of noise


(c) Napkin x7

(d) Absolute difference.

Figure 7.10: Boats denoised using the Napkin modeling scheme.



(a) Clean

(b) 30% of noise

(c) Napkin ×7.

(d) Absolute difference.

Figure 7.11: Barb denoised using the Napkin modeling scheme.

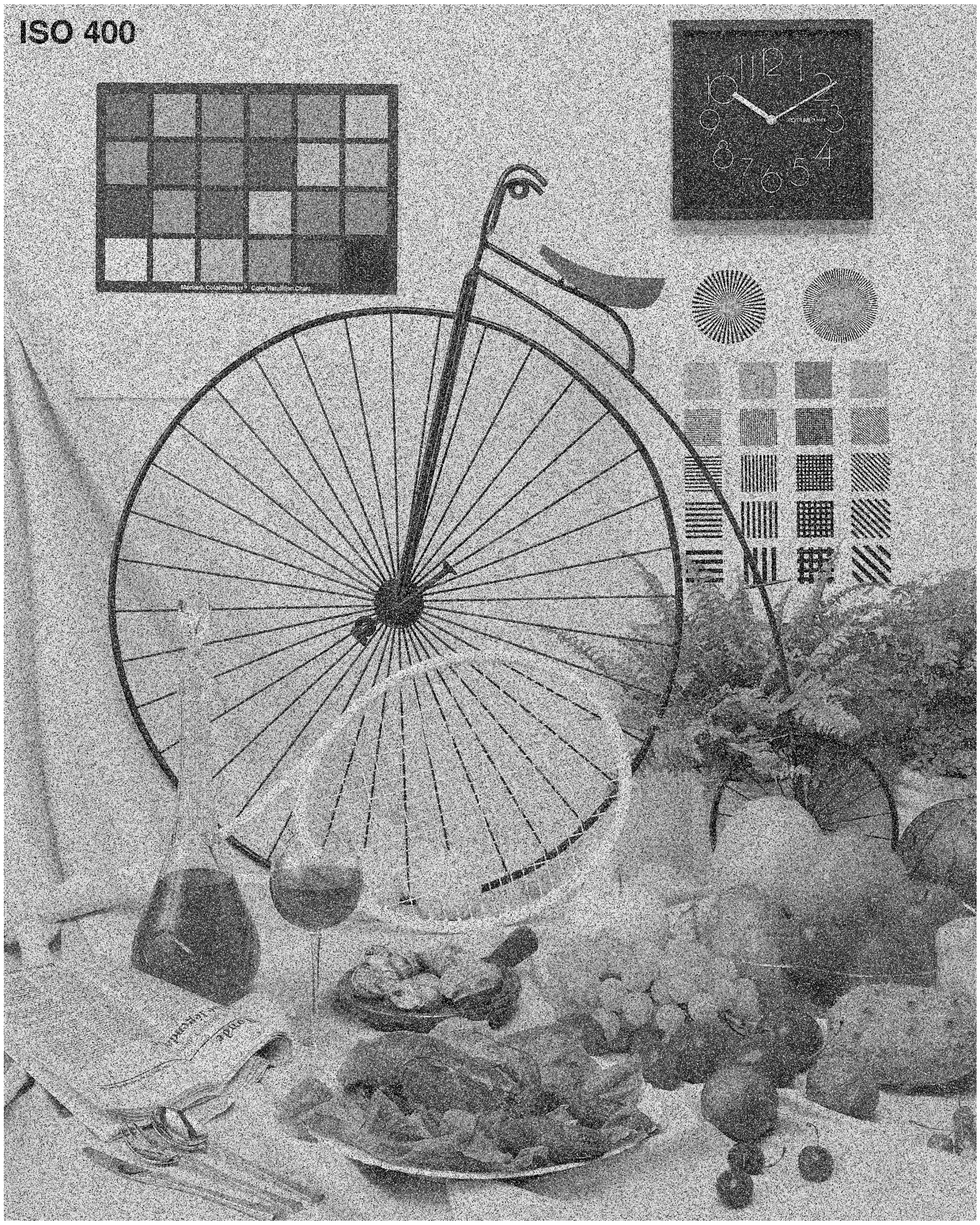


Figure 7.12: Bike under Salt and Pepper noise with $\lambda = 30\%$

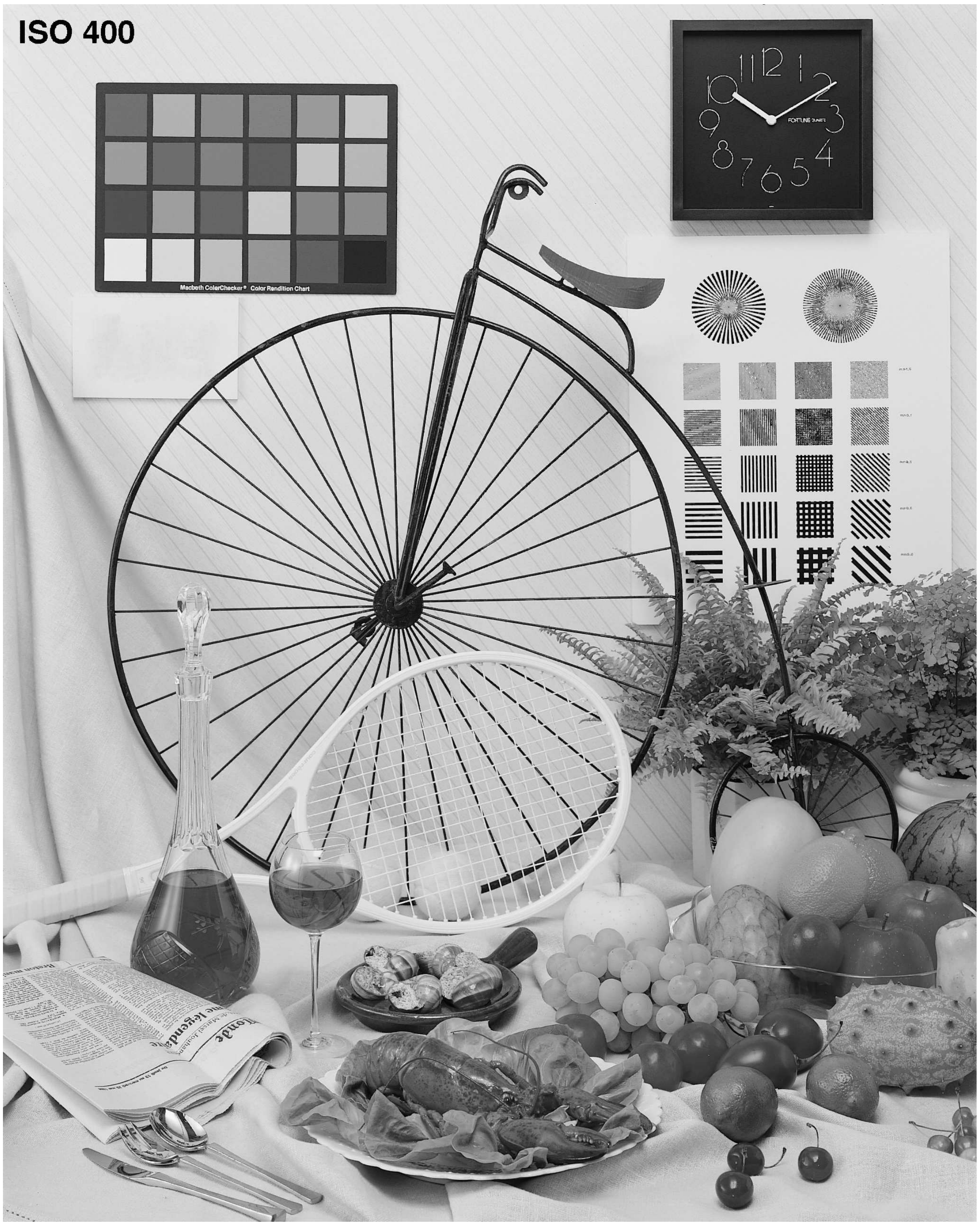


Figure 7.13: Bike denoised. PSNR=29.5dB.

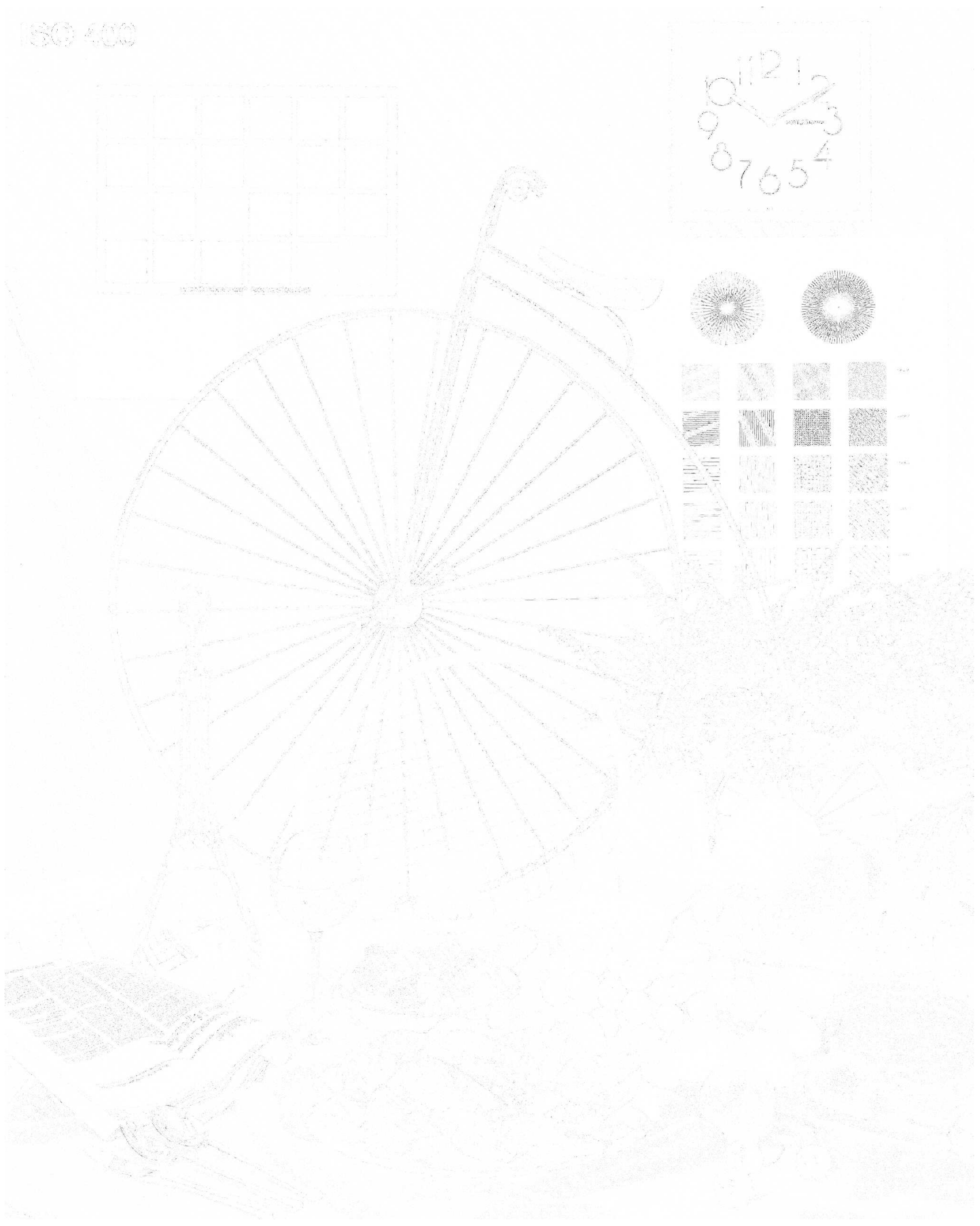


Figure 7.14: Absolute difference between clean bike and denoised bike.

### Selection of the parameters

The Napkin modeler has many parameters. Only the ones that have shown a greater impact on the performance are shown. As for the Legacy case, the number of context classes has a different optimal value for each different image size. We show the parameter selection results for two image sizes: the four small images (Boats, Barb, Bridge and Lena, of about $0.25MP$) and the large Bike (of $4MP$ pixels, which is the size of the images produced by the current digital cameras of many of the images in the JPEG-LS test suite).

**Context class features** The context modeling stage (described in Section 6.6.3) produces a set of features which describe aspects of the context centered at each pixel. Each of these features can be described with a selectable number of bits (including 0) and the result is then concatenated to build the final context class. This can be done independently for the probability context class and the adaptive prediction context class. The following graphs show how the fixed part of the predictor behaves with respect to each one of these features. For this, the adaptive bias cancellation was disabled and the distribution of the prediction residuals was studied for each class when the probability context classes were solely determined by one feature at a time. Figures 7.15 to 7.18 show conditional prediction error distributions when the context classes are determined from each feature.

The results show a clear dependency with the activity level and gradient angle component. The first affects mostly the shape of the distribution while the second has a stronger effect on the bias, with a lesser effect on the shape of the distribution. The texture element, although less clear, also has an important influence on the bias term. The results with the different combinations have shown that a good combination is to use activity level as the “base” feature used both in distribution conditioning and bias cancellation, with an added texture bits signature for the bias cancellation terms. The gradient angle has not shown to be as relevant as could have been expected, and remains as a subject for further experimentation.

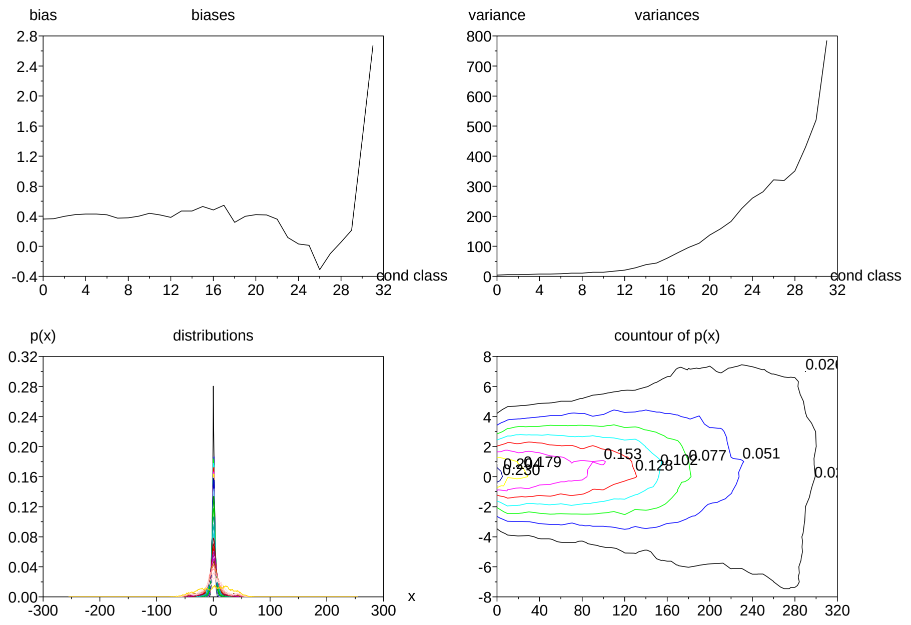


Figure 7.15: Prediction error distributions conditioned on 32 quantized activity levels.

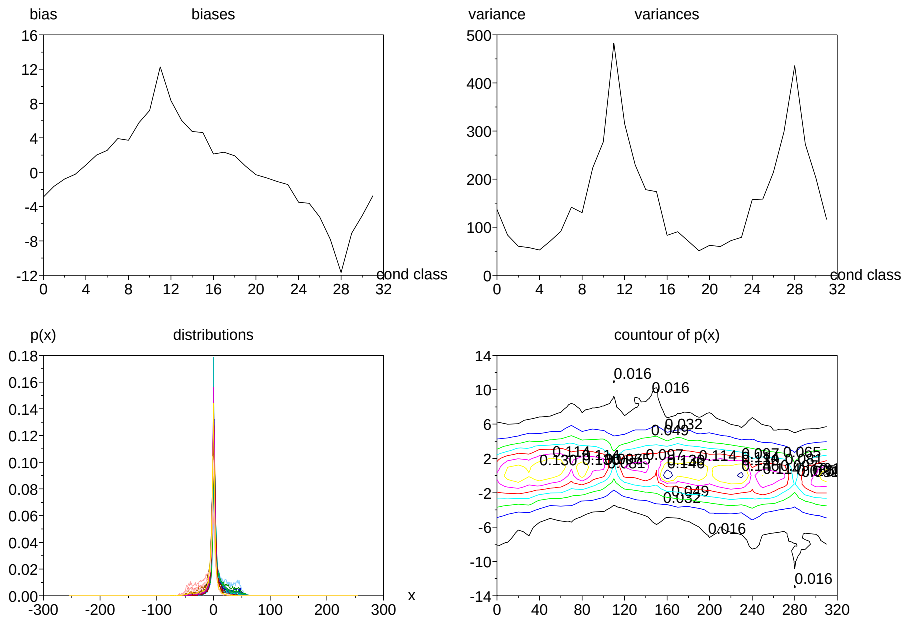


Figure 7.16: Prediction error distributions conditioned on 32 quantized gradient angles.

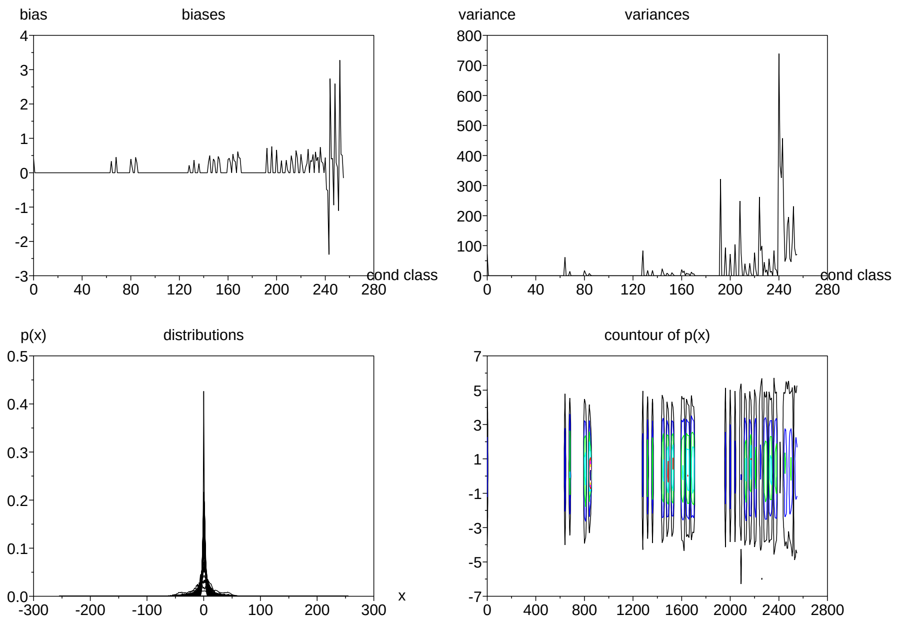


Figure 7.17: Prediction error distributions conditioned on 4x64 wing gradient magnitudes. Because of the canonical mapping, not all combinations actually appear.

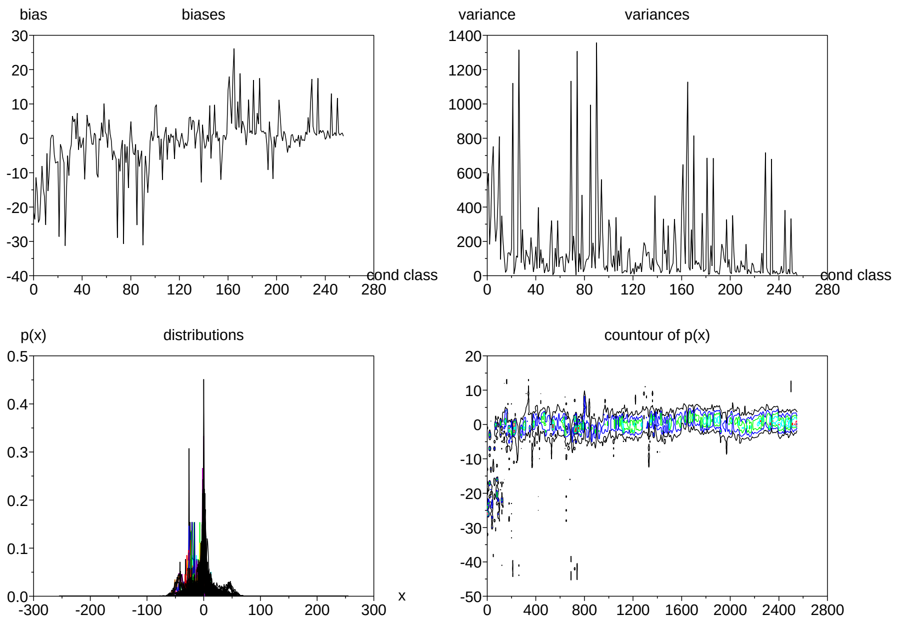


Figure 7.18: Prediction error distributions conditioned on 8-bit (256 possible) texture bitmaps.

| pred. | barb | boats | bridge | lena |
|---|---|---|---|---|
| average | 4.4 | 3.7 | 5.1 | 3.8 |
| sharp | 4.5 | 3.8 | 5.2 | 3.9 |
| smooth | 4.5 | 3.8 | 5.3 | 3.9 |

| pred. | barb | boats | bridge | lena |
|---|---|---|---|---|
| average | 29.4 | 35.1 | 29.1 | 35.7 |
| sharp | 29.1 | 33.9 | 28.7 | 34.6 |
| smooth | 28.7 | 33.3 | 28.1 | 34.5 |

Table 7.5: Performance of Napkin vs. prediction filter. Left: Compressibility (average bits per pixel); Right:Denoising performance (PSNR in dB)

**Number of conditioning classes** Figure 7.19 shows how the denoiser performance varies when the activity level bits are increased for both the distribution conditioning classes and the adaptive predictor conditioning classes.

An intersting result from this experiment is the clear dependency between optimal number of conditioning classes and image size. Observe that the Bike image is better denoised with a high number of states, while the rest see their performance dropped after approximately 8 states. This behavior also holds for the rest of the JPEG-LS suite (the bigger images).

Now, Figure 7.20 shows a slightly different experiment in which the distribution conditioning classes are fixed for the adaptive prediction, and vary from 0 (no distribution conditioning, only one context) to 3. Surprisingly, this parameter has no effect on the overall performance. Thus, the real improvement lies in the prediction part, while the distribution conditioning which is the base of the DUDE algorithm has no effect on the final result.

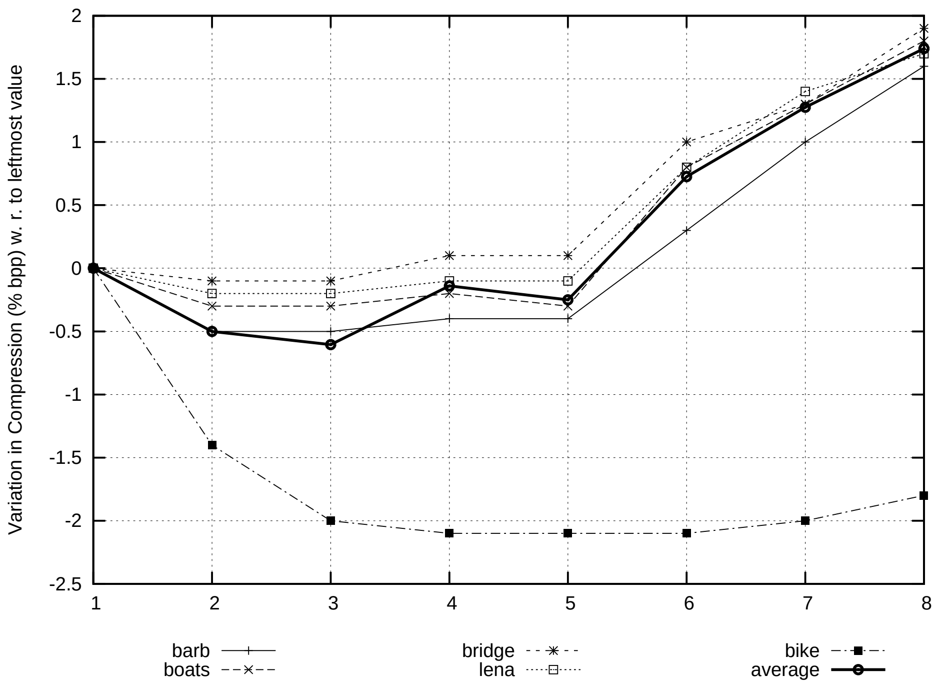
This result can be explained by examining the form of the denoiser for the impulse channel for the noisy cases, (6.50) and (6.51). The decision of the DUDE for each noisy pixel is practically that of substituting it with the average of the distribution of the error prediction centered at the predicted value, which in turn yields a value very close to the predicted value itself. In this framework, the context modeling plays the role of refining the overall prediction by letting many different predictors work specifically for a set of similar contexts (those of the same class). When these contexts are indeed similar in terms of predictor behavior, the context modeling increases the denoising performance.

**Canonical mapping, DC offset removal** The results for the Legacy modeling also apply to this case, with the same (relative) results.

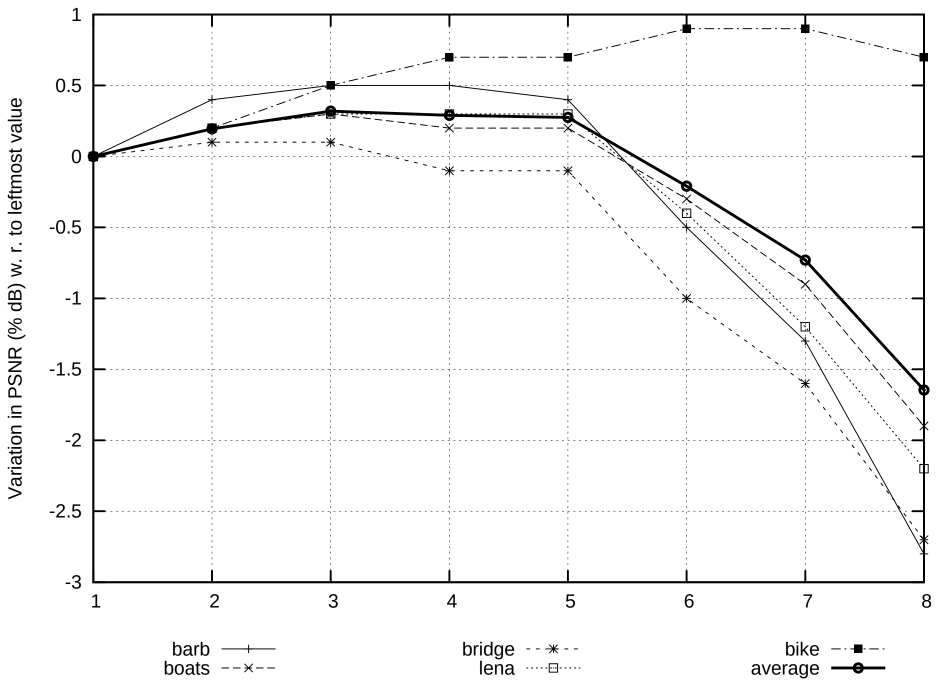
**Predictor variant** The fixed part of the Napkin has three variants described in Section 6.6.4: the Average Variant, the Sharp Variant and the Smooth Variant. The Smooth variant, in particular, was designed to be more robust to additive noise. In any case, the three variants were tested with each type of noise (additive and non-additive). The results are detailed in Table 7.5.

**Gradient threshold** Figure 7.21 shows the experiment which led to the selection of 8% as the optimal value for the overall case.

This is a rather nonintuitive parameter. Basically, it controls the sensitivity of the Napkin predictor. A low value makes the Napkin consider more wings as nonflat, thus making it work more like and edge detector (see 6.6.4 for details). On the converse, a higher value will make it behave more like a window average filter.



(a) Compressibility.



(b) Fidelity.

Figure 7.19: Performance measures for different number of conditioning classes.

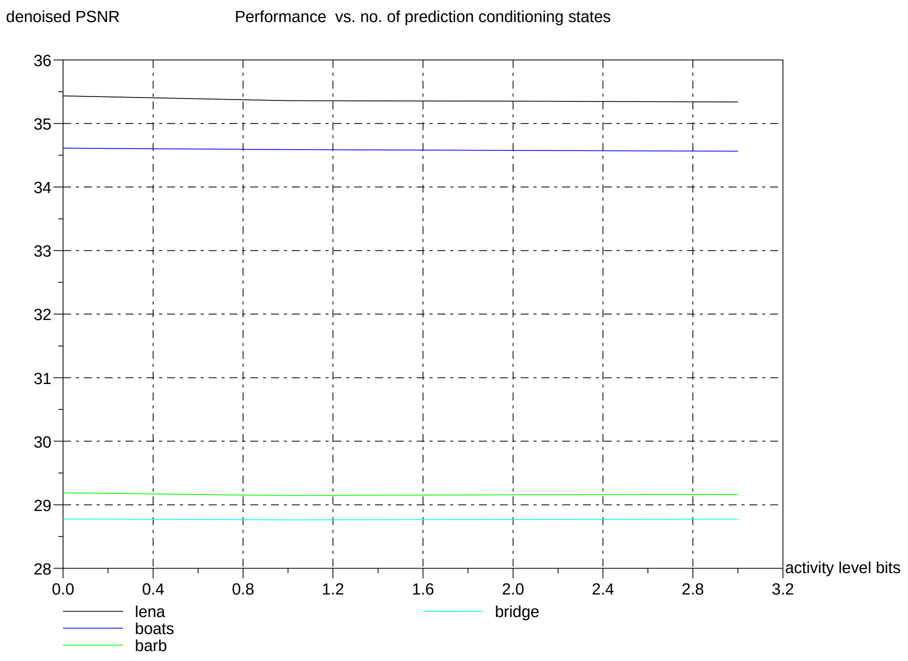


Figure 7.20: Performance measures for different number of adaptive prediction conditioning classes (results only available for this subset)

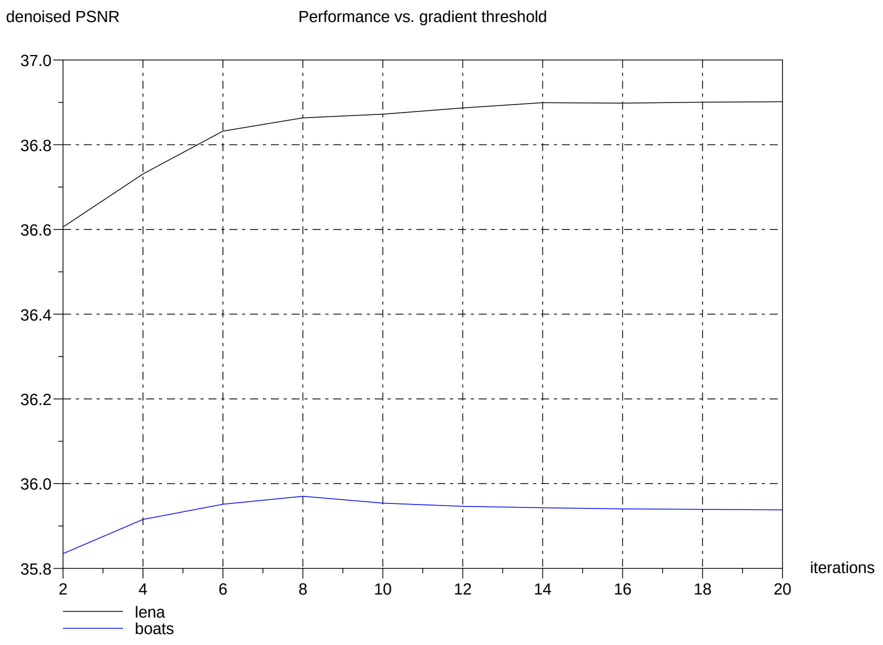
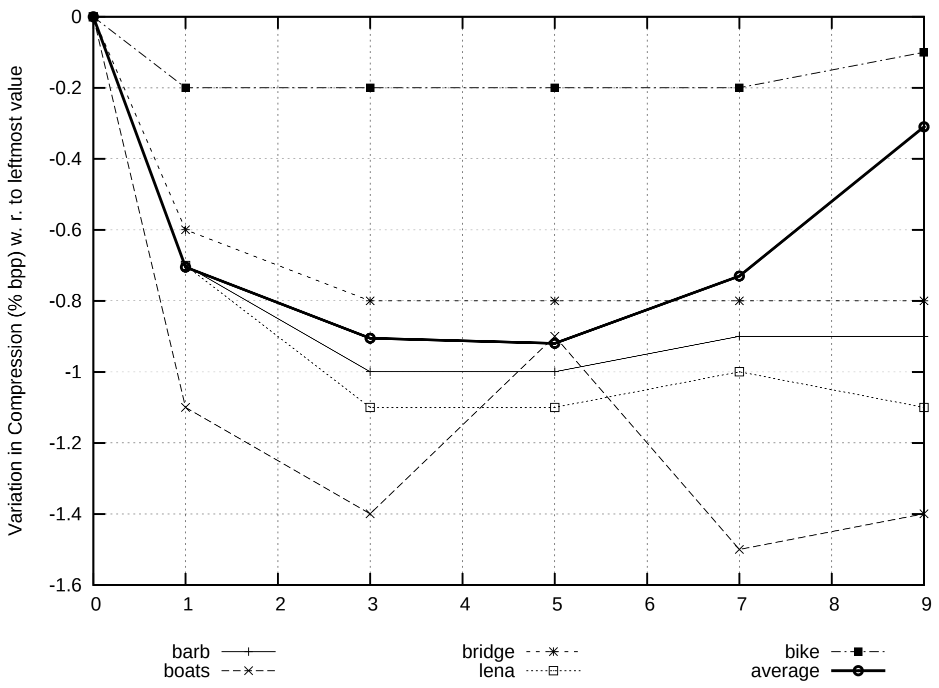
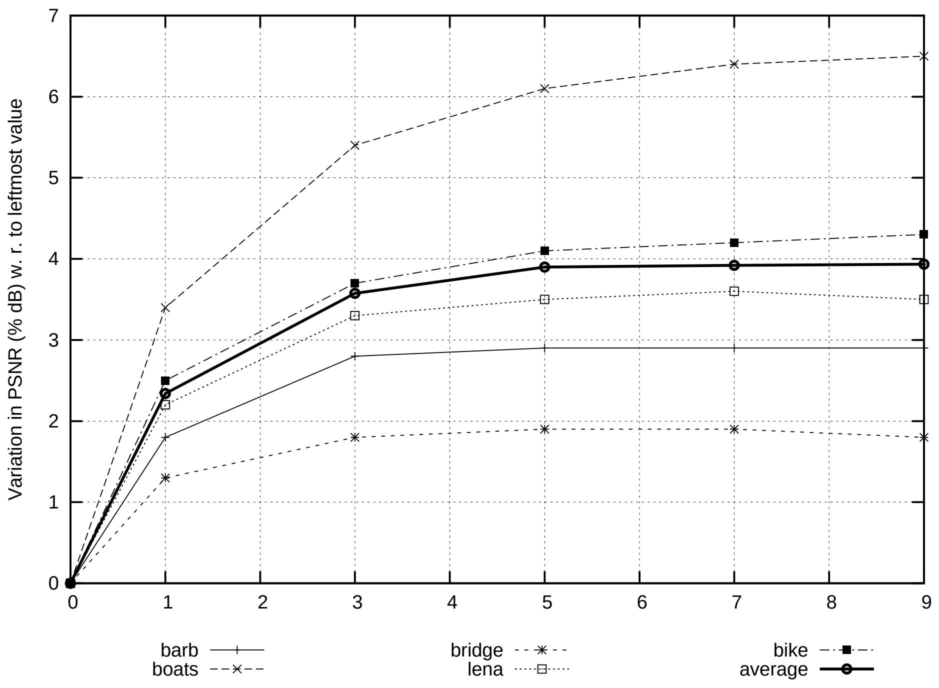


Figure 7.21: Napkin performance vs. gradient threshold.

**Recursive prefiltering applications** Figure 7.22 shows the recursive behavior of the DUDE-I when using the Napkin modeling scheme. A first remark is that its performence does not reach a saturation as soon as the Legacy modeling. The algorithm is also significantly faster and thus recursion is not an expensive operation in this case. This will be shown to be very important to achieve good results under higher noise rates.



(a) Compressibility.



(b) Fidelity.

Figure 7.22: Napkin performance vs. number of recursive prefiltering applications for $\lambda = 30\%$.

| image | reference | Napkin (x60) |
|---|---|---|
| lena | 29.3 | **30.7** |
| boats | 16.7 | **30.3** |
| bridge | 25.0 | 24.5 |
| barb | 16.0 | **24.9** |

Table 7.6: Napkin results for the case $\lambda = 70\%$.

| image | MND-DP | Napkin (x60) |
|---|---|---|
| lena | **25.4** | 22.4 |
| boats | - | 20.4 |
| bridge | **21.5** | 20.1 |
| barb | - | 20.3 |

Table 7.7: Best Napkin results for the case $\lambda = 90\%$ compared to the MND-DP algorithm.

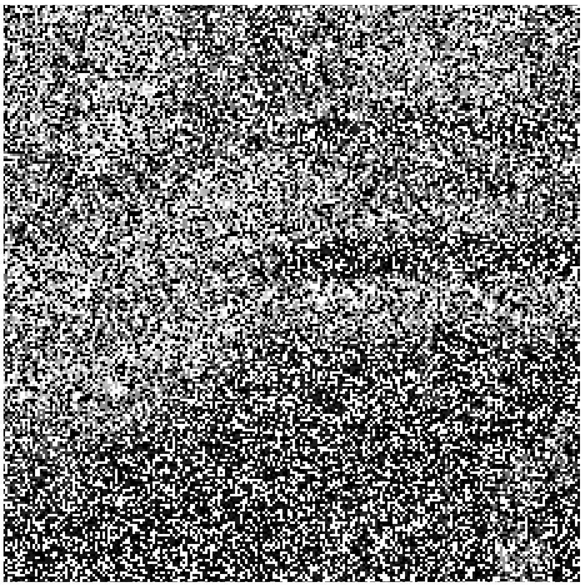
### Extreme Salt and Pepper: very high probability of noise

The case $\lambda > 50\%$ is an interesting setting and special algorithms have been developed for it. The best results of the proposed solution when $\lambda = 70\%$ and $\lambda = 90\%$ are compared with the results in [2] as a reference. For this case, the recursive prefiltering scheme was applied up to 60 times. The rest of the parameters are the same as the previous results.

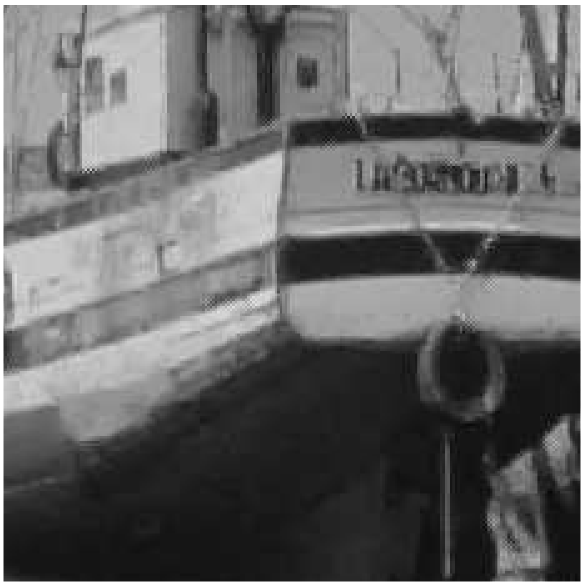
Figures 7.23 and 7.24 show how the denoised Boats and Barb images look for the case when $\lambda = 70\%$, while Figure 7.25 shows the result for Lena corrupted by $\lambda = 90\%$ of noise.

The results are also good for this extreme setting. For $\lambda = 70\%$, the state of the art for the Bridge image is matched, and the results for Lena are improved with only an increase in the number of recursive prefiltering applications (which could be determined automatically using the compressibility criterion). Finally, while the results for $\lambda = 90\%$ do not reach the state-of-the-art, they are obtained using our tool, which is more flexible and generic than [2], an algorithm that is aimed specifically at this type of noise in this extreme setting.

(a) Clean



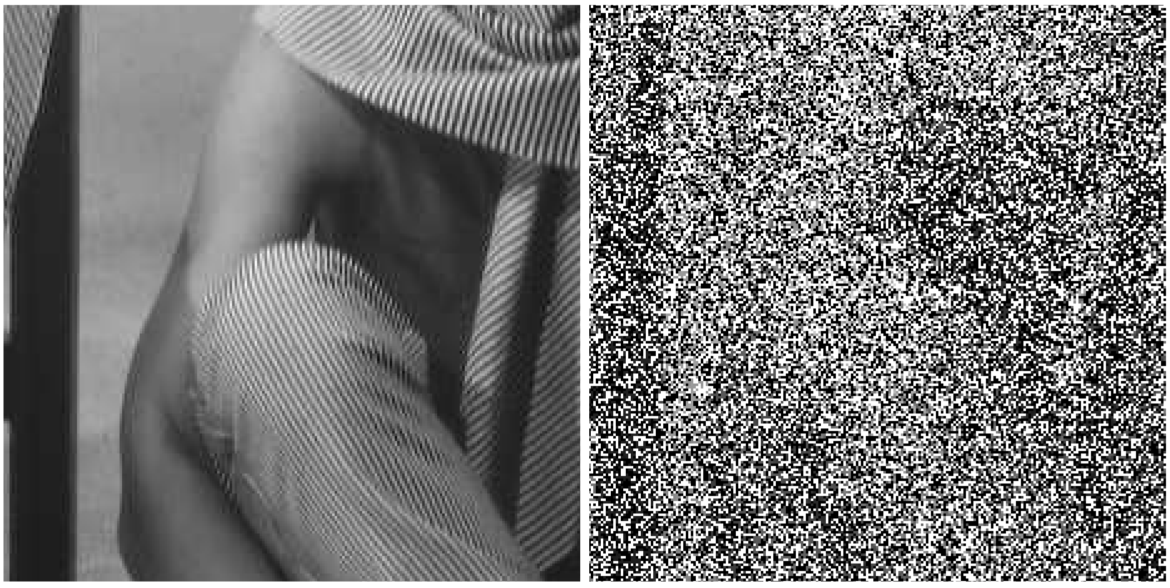
(b) 70% of noise.



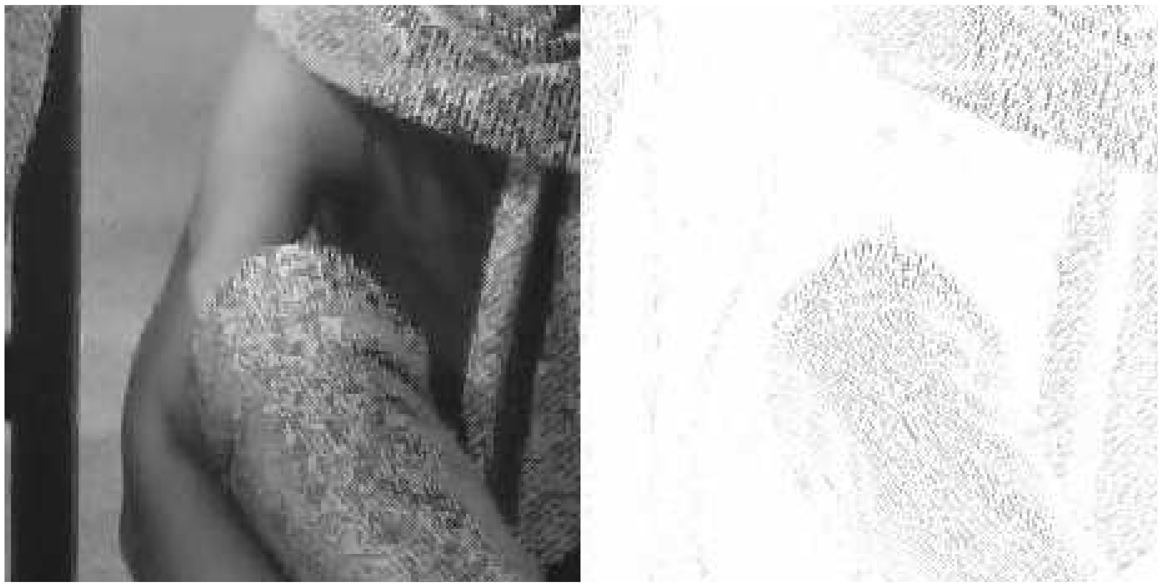
(c) Napkin x60.

(d) Absolute difference.

Figure 7.23: Results for Boats corrupted by S&P with $\lambda = 70$

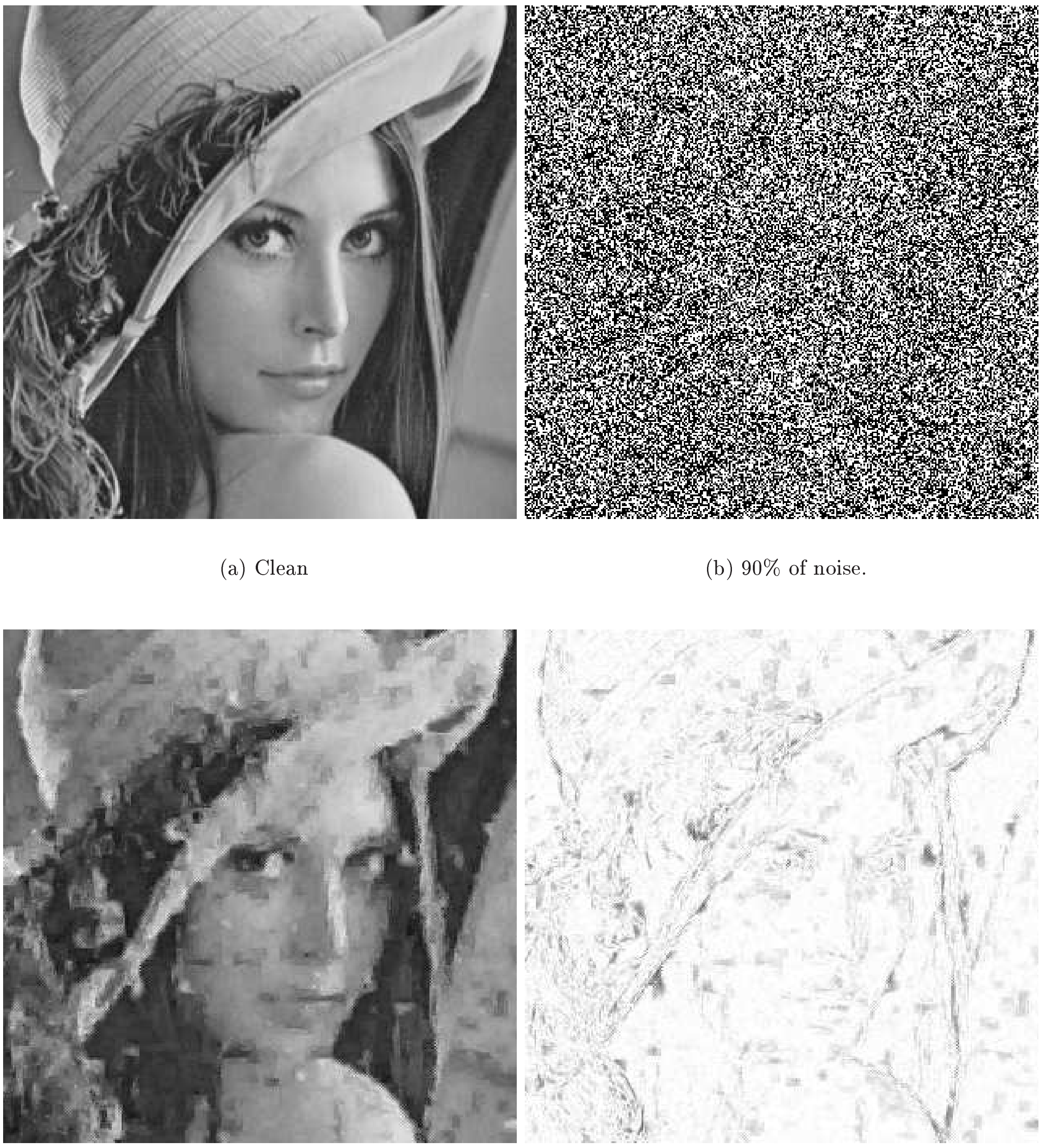


(a) Clean (b) 70% of noise



(c) Napkin x60 (d) Absolute difference.

Figure 7.24: Results for Barb corrupted by S&P with $\lambda = 70$



(a) Clean

(b) 90% of noise.

(c) Napkin x60.

(d) Absolute difference with clean.

Figure 7.25: Results for Lena corrupted by S&P with $\lambda = 90\%$

| image | reference | Legacy (x2) | Napkin (x7) | Combined (x5) |
|---|---|---|---|---|
| lena | 34.3 | 37.7 | **38.3** | 37.8 |
| boats | 32.1 | 36.8 | 38.3 | **38.5** |
| bridge | **31.5** | 30.0 | 30.6 | 30.7 |
| barb | 28.1 | 33.7 | 32.2 | **34.7** |

Table 7.8: Combined LBG+Napkin

### 7.2.4 Combined LBG+Napkin

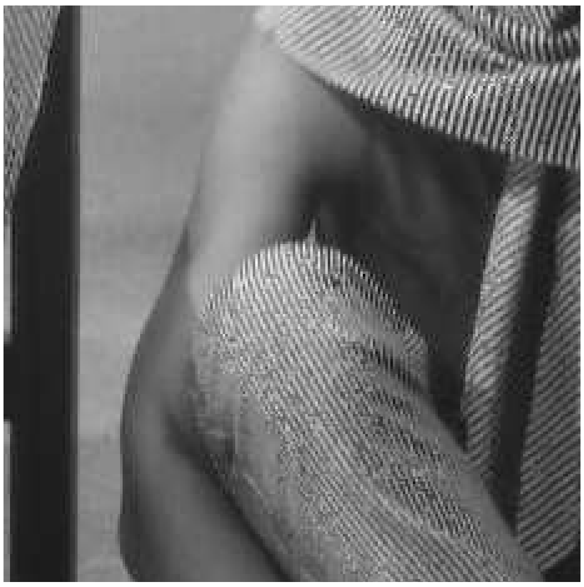
While the results for the Napkin scheme surpassed the Legacy results in the ample majority of the cases (from the full test suite), they fell short of Legacy for the two versions of "Barb". The visual inspection of the denoised Barb for both methods revealed some notorious errors produced by the Napkin model in certain regions of the image. To isolate the problem, and given that the prediction and context modeling parts of both modeling schemes could be interchanged, a *Combined LBG/Napkin scheme* was implemented where the modeling part was the LBG used in the Legacy Scheme (Section 6.5.4), and the predictor was the Average Variant used in the Napkin Scheme (Section 6.6.4).

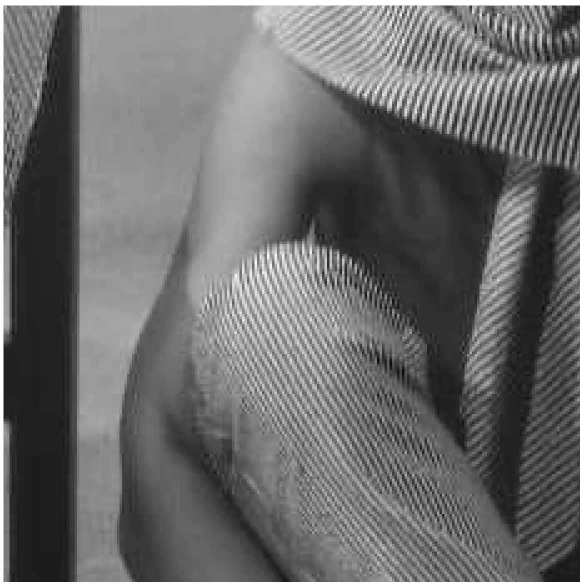
In this modeling scheme, the **fixed** prediction part of the Napkin Modeling, namely the Napkin filter, was used as the predictor filter of the Legacy scheme to yield the results of Table 7.8. With this modification , the number of recursive prefiltering applications rose to 5 before reaching a saturation point.

Of the results in table 7.8, which give an overall advantage to the Combined Scheme, the difference between the two modeling approaches is very important for the Barb image.Indeed, the results for this case are better than the Legacy results. Thus, the problem of the Napkin Scheme with Barb lies in the context modeling part. Figure 7.26 shows a detail of both denoised images in which the source of the difference is clearly seen: the Napkin modeler was jittered by the highly changing sections of the image located mostly at the stripes and checkers around the image. This in turn affected the conditional distributions which grew too wide favouring the noise patterns and thus decreased the overall performance in those areas.

### 7.2.5 Comparison of the modeling approaches

Up to now, one could say that the best modeling scheme is the one which combines LBG with Napkin prediction. However, when computational resources are important, especially execution time, the best tradeoff is obtained with the Napkin Modeling scheme. Table 7.9 shows the time consumed in the first pass (modeling) and the second pass (denoising) for each modeling scheme and a series of images of different size. Even the largest one is below 1 MP (megapixel), the lowest resolution any digital camera can take pictures at. The required memory and computational time required to denoise a 4 MP image (bike) with the Legacy modeling was simply too much for the machine in which these tests were performed (Pentium 4 at 2.2 GHz, 1 GB of RAM, compiled with the GNU C++ Compiler V3.3 at maximum optimization). Even when dealing with the smaller images, the higher noise cases ($\lambda = 50\%$ or $\lambda = 70\%$) could not be attacked with this scheme because of the large number of recursive applications needed.


(a) Napkin x7


(b) Combo x5

(c) Absolute difference:Napkin x7

(d) Absolute difference:Combo x5

Figure 7.26: Detail of barb as denoised by Napkin and Combo. The difference is clearly observed in the stripes all over the image.

| size (pixels) | LBG (s) | Napkin (s) | 2nd. pass.(s) |
|---|---|---|---|
| 95052 | 57.0 | 0.7 | 0.7 |
| 190134 | 113.0 | 1.5 | 1.5 |
| 380208 | 223.0 | 3.0 | 3.0 |
| 760285 | 441.0 | 6.0 | 6.0 |

Table 7.9: Execution time vs. image size for the different modeling schemes in the first pass and for the second pass. The number of conditioning classes is 256 for the LBG Scheme and 8 for the Napkin Scheme.

### 7.2.6 Sensibility to pseudo-random noise generation

The above results use the same simmulated noisy images for each noise type and parameter. A final validation that applies to all of the above schemes is to study how much do the results vary with different samples of the noise simulated by the pseudo-random numbre generator (different initial random seeds in the pseudo-random number generation function). Table 7.10 shows show this analysis for the common setting of $\lambda = 30\%$, for 16 different random seeds and for two different number of recursions. The modeling scheme used for this test is Napkin, due to the large number of tests required.

| recursions | image | min | max | mean | std. dev. |
|---|---|---|---|---|---|
| 0 | lena | 36.74 | 37.11 | 36.92 | 0.11 |
| | boats | 35.79 | 36.20 | 36.00 | 0.12 |
| | barb | 30.74 | 30.82 | 30.78 | 0.04 |
| | bridge | 29.80 | 30.05 | 29.96 | 0.07 |
| 1 | lena | 37.49 | 37.96 | 37.70 | 0.14 |
| | boats | 36.95 | 37.41 | 37.24 | 0.12 |
| | barb | 31.25 | 31.46 | 31.37 | 0.05 |
| | bridge | 30.19 | 30.43 | 30.34 | 0.06 |
| 4 | lena | 37.81 | 38.27 | 38.10 | 0.14 |
| | boats | 37.87 | 38.41 | 38.20 | 0.13 |
| | barb | 31.71 | 31.94 | 31.83 | 0.06 |
| | bridge | 30.36 | 30.61 | 30.52 | 0.06 |

Table 7.10: Sensibility of the result for 16 different random noise simmulations. These results are obtained using the Napkin modeling scheme.

The first interesting result is that the standard deviation is roughly independent of the number of iterations. This gives some sort of "stability" measure for the recursive denoising process. On the overall, it is seen that a variation of around $0.1dB$ is not significant in any of the experimental results for the impulse noise and this modeling scheme.

#### Preclassification of impulse noise

When possible, preclassification is a valuable tool. However, it is a difficult tool to use, mostly when used to perform a selective denoising, as a *miss* (i.e., to mark a noisy pixel as clean) could lead to very noticeable noisy pixels left untouched. On the other side, when used only as an aid to the modeling stage, for instance to avoid jittering in the bias cancellation term adaptation that apperars in the Napkin predictor, it can improve the overall performance significantly.

The following experiment, whose results are shown in Table 7.11 is a side test that focuses only on the preclassification performance for the Salt and Pepper case. As mentioned before, the best approach in this case is to use the Trivial preclassification scheme described in Section 6.2.1, which has a high number of *false hits* (i.e., clean pixels marked as noisy) but no misses. The homogeneity preclassification, however, will prove to be a valuable method when confronted with more difficult noise types such as the $q$-ary symmetric described in Section 7.3.

| image | noise | trivial | | homogeneity | | DUDE | |
|---|---|---|---|---|---|---|---|
| lena | $\lambda$ | false | misses | false | misses | false | misses |
| | 10% | 0 | 0 | 23 | 0 | 0 | 0 |
| | 20% | 0 | 0 | 32 | 8 | 0 | 0 |
| | 30% | 0 | 0 | 39 | 29 | 0 | 0 |
| bridge | 10% | 97 | 0 | 35 | 47 | 122 | 37 |
| | 20% | 92 | 0 | 42 | 84 | 108 | 88 |
| | 30% | 86 | 0 | 49 | 127 | 146 | 109 |

Table 7.11: Preclassification results for the different approaches.

Note that Lena does not have any black or white pixels and thus the trivial classification is perfect in this case. This explains the poor performance of the CSAM algorithm (which is more "fair") with Lena when compared even to a selective median (which is based also on this trivial preclassification scheme). As the number of false hits in Bridge corresponds to the white and black regions which are always the same, this number can only decrease for the trivial classifier, as more of those pixels will eventually be corrupted (although with the same resulting value).[5]

### Sensibility to the channel parameter

Up to now the DUDE-I had perfect knowledge of the channel paramenters. The purpose of the following experiments is to see how the performance is affected when the channel parameter is not the correct one but lies within a range centered at the true parameter. Table 7.27 shows the case where the true $\lambda = 30\%$ and the estimated parameter $\lambda'$ varies from 20 to 60.

The experiment shows a high sensitivity to values lower than the true parameter, but virtually no impact for higher ones. This result can also be explained by inspecting (6.50) or (6.51) in Section 6.7.4. A value of $\lambda'$ greater than the true $\lambda$ will yield negative distribution values at 0 and $M-1$. When the distribution is later corrected to be a valid probability distribution, this will only have a scale effect which does not affect the denoising function. On the converse, a value of $\lambda'$ smaller than $\lambda$ will leave nonzero residuals in distribution at those points which will disturb the following calculations in a noticeable way (as the denoising function results in the average of the final distribution, two similar peaks at 0 and $M-1$ will move the result towards $M/2$).

## 7.2.7 Asymmetric impulse and the Z-Channel

The purpose of this experiment is to study how the asymmetry of the impulse channel affects the denoising performance. The impact should affect mostly the prefiltering stage, as more

[5]The false hits and misses counts are computed when comparing the resulting masks with the true noise masks, which are computed along with the noisy image and thus the concept of "noisy" pixel includes every pixel that is touched, even if it ends up with the same value it had before going through the channel.

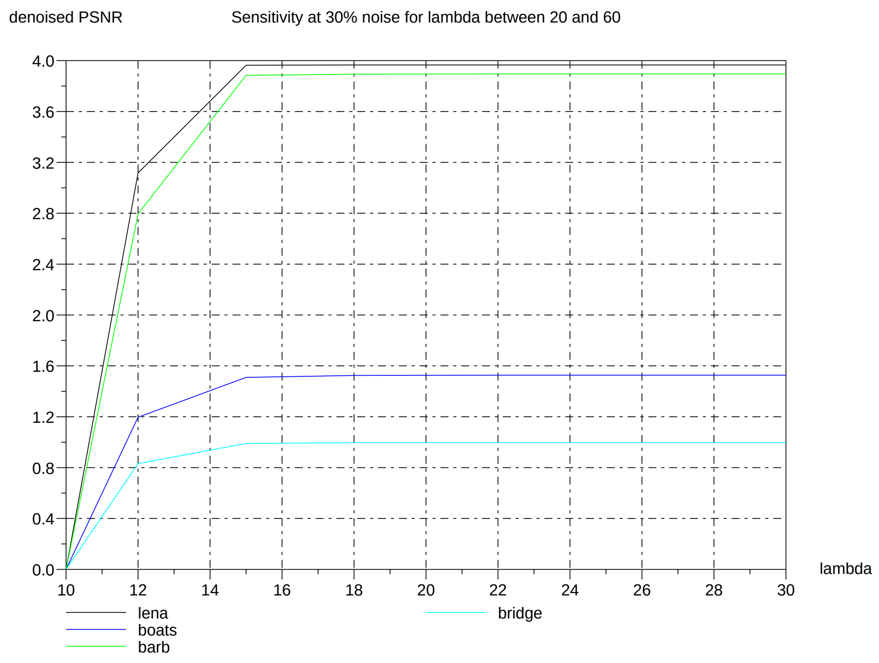


Figure 7.27: Performance vs. $\lambda'$ for $\lambda = 30\%$.

asymmetric channels will favour bursts of one of the impulse noise values ("salt" or "pepper") which could confuse the median filter.

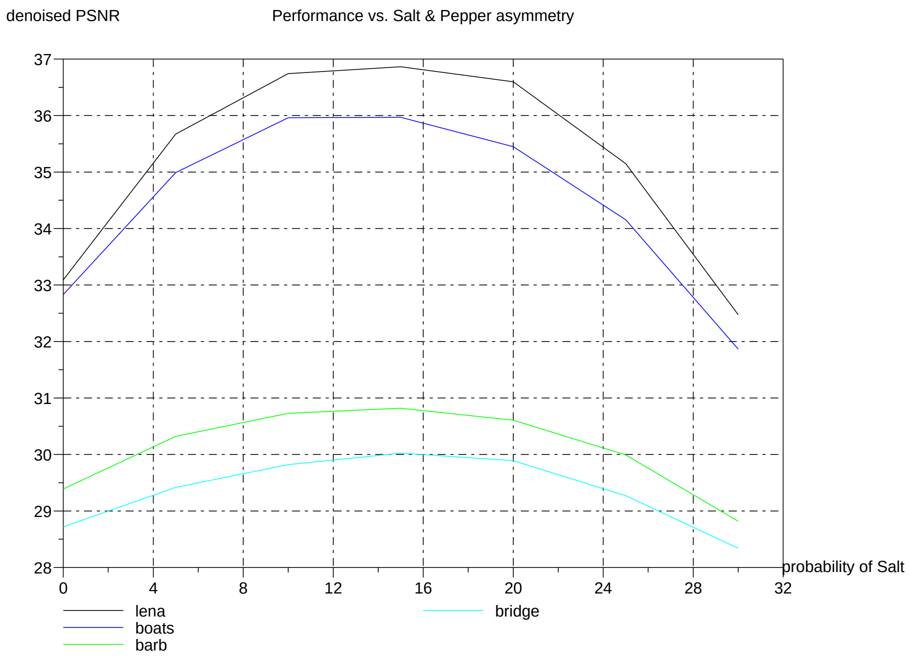
The preclassification should not be affected in the average, as both noise values are equally non additive and will appear equally "strange" within their contexts.

To focus on the asymmetry, the overall probability of error is fixed to $\lambda = 30\%$ and let the probability of "salt" range from 0 to 30% (thus "pepper" will range from 30% to 0). The results are shown in Figure 7.28(a) for a nonrecursive prefiltering approach.

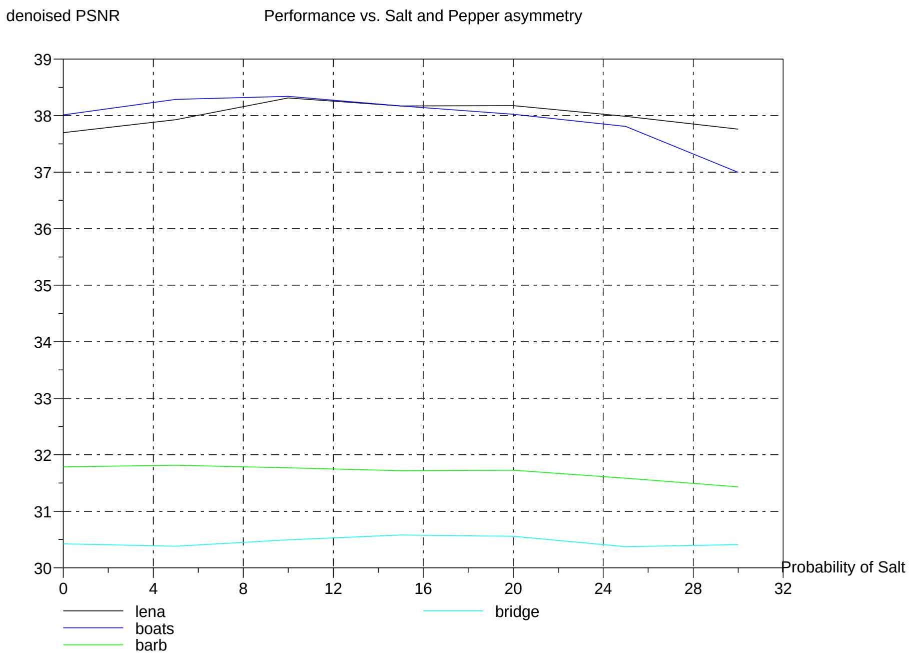
A quick examination of Figure 7.28(a) shows that the result is worst for "all salt" than for "all pepper". This is due to the fact that the clean test images are closer to black than to white, and a nonrecursive execution will leave "all salt" bursts which decrease the PSNR. As the recursive prefiltering gradually removes the bursts, the result should be more and more symmetric. This is verified in Figure 7.28(b).

## 7.3 $q$-ary symmetric channel

This channel is more difficult than the other non-additive noise models presented so far, as the corrupted samples can take any value. The flexibility of the DUDE-I (and of the baseline DUDE scheme) is demonstrated in this case, which so far has not treated by other methods in the literature. Table 7.12 shows the different denoising performances for various noise levels. As there are no reference results from other works, the the results of applying a simple median filter are used as a reference. The DUDE-I was configured to use the homogeneity preclassification



(a) Nonrecursive DUDE-I



(b) Recursive DUDE-I (x7)

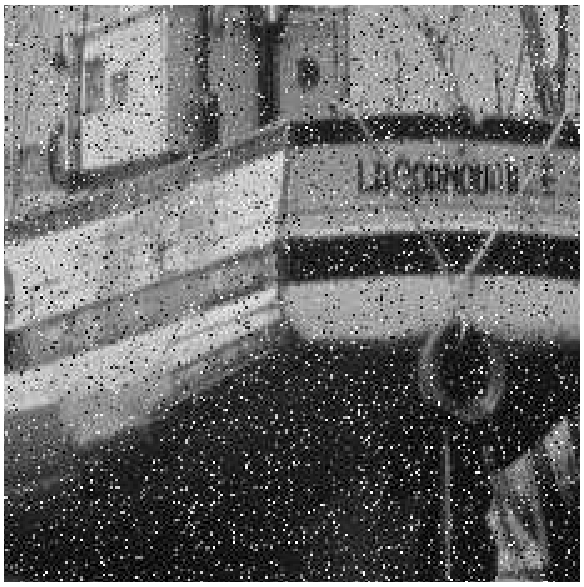
Figure 7.28: Napkin results for the asymmetric impulse channel.

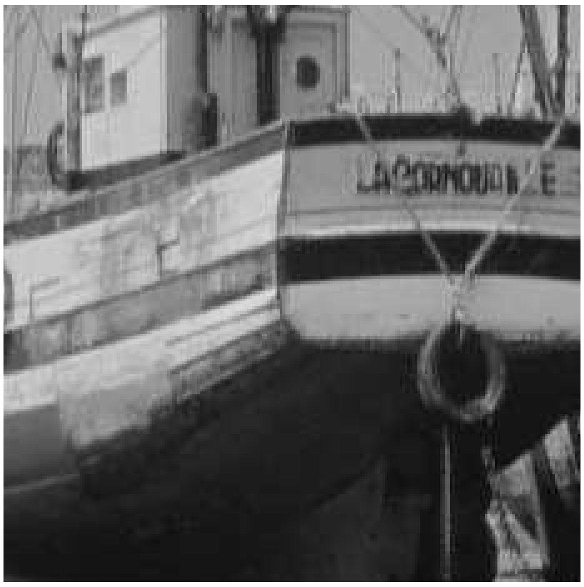
scheme but only for bias cancellation adaptation purposes. Figure 7.29 shows a sample denoised image for this type of noise.

| image | 10% | | 20% | | 30% | |
|---|---|---|---|---|---|---|
| | median | DUDE-I | median | DUDE-I | median | DUDE-I |
| lena | 30.0 | **37.0** | 29.3 | **34.2** | 28.3 | **31.8** |
| boats | 28.5 | **36.3** | 27.7 | **33.0** | 26.8 | **30.6** |
| barb | 23.5 | **31.4** | 23.2 | **28.4** | 22.8 | **26.4** |
| bridge | 23.4 | **30.6** | 23.0 | **28.0** | 22.4 | **26.3** |

Table 7.12: Denoising performance for the $q$-ary symmetric channel and different probabilities of error.

(a) Clean


(b) Noisy


(c) Napkin x4

(d) Absolute difference.

Figure 7.29: Results for Boats corrupted by q-ary symmetric channel with $p_{err} = 10\%$

| $\sigma$ | image | noisy | Wiener | GIO | WCC | SMG | NLM | FOE |
|---|---|---|---|---|---|---|---|---|
| 10 | lena | 28.1 | 33.6 | 34.0 | - | **35.6** | - | 35.0 |
| | boats | 28.1 | 33.2 | 33.7 | - | - | - | **33.0** |
| | barb | 28.1 | 31.5 | 32.0 | - | **34.0** | - | 32.8 |
| 20 | lena | 22.1 | 30.0 | 30.6 | **32.7** | 32.7 | 29.9 | 31.9 |
| | boats | 22.2 | 29.4 | 30.2 | - | - | - | **29.9** |
| | barb | 22.2 | 27.2 | 27.9 | - | **30.3** | - | 28.3 |
| 25 | lena | 20.2 | 28.9 | 29.4 | - | **31.7** | - | 30.8 |
| | boats | 20.4 | 28.3 | 29.1 | - | **30.8** | - | 28.7 |
| | barb | 20.3 | 26.0 | 26.6 | - | 29.1 | **29.6** | 27.0 |

Table 7.13: Reference results for the Gaussian channel.

## 7.4 Gaussian channel

The Gaussian noise is usually studied with a standard deviation $\sigma$ that goes from 5 to 25 in the 8 bit, 256 graylevel scale, with $\sigma = 20$ being the typical setting for "high" noise.

### 7.4.1 Reference

Table 7.13 shows the current state of the art in gaussian denoising. The algorithms are:

**Wiener** Wiener filter (as defined by the `wiener2` function of MatLab $^{\text{TM}}$) applied to $5 \times 5$ square windows.

**WCC** Wavelet-Curvelet Combination [30] .

**SMG** Scale Mixtures of Gaussians [25] .

**NLM** Non Local Means [1] .

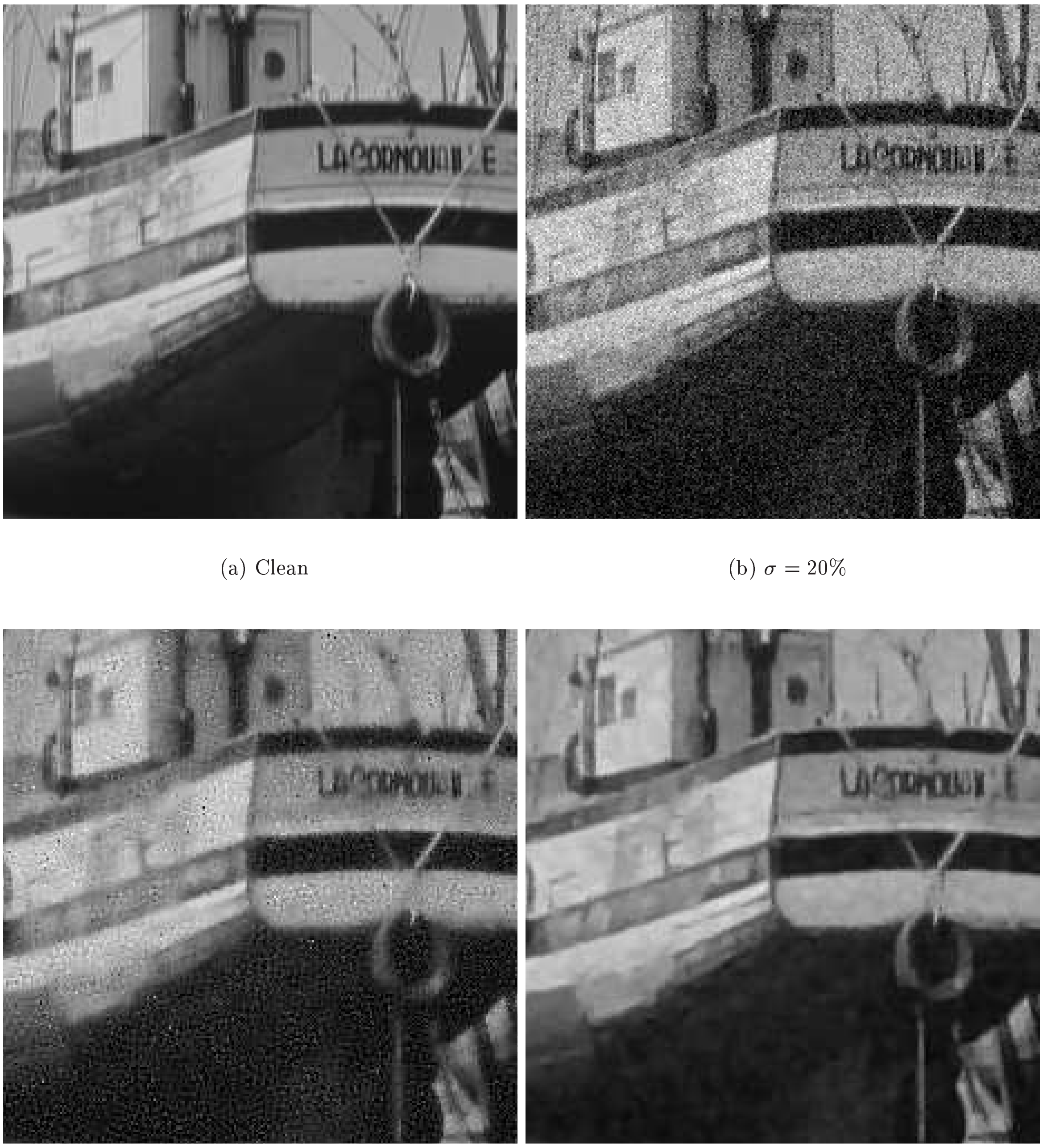
**FOE** Field of Experts [27] .

**GIO** DUDE adaptation to continuous tone images, previous version [18].

### 7.4.2 Legacy

Table 7.14 shows the best results for the Legacy modeling scheme compared to the best values of Table 7.13, while Figure 7.30 shows a sample result. The best parameters for this case were found to be the following:

- 128 conditioning classes.
- The clusters are obtained after 25 LBG iterations.
- Prefitering is *not* performed. The noisy input is used as is to produce the conditioning classes.
- Prediction is performed using an Average filter over $5 \times 5$ windows.
- The square error function ($L_2$) is used as the loss model.



(a) Clean (b) $\sigma = 20\%$

(c) Wiener (d) Legacy (x1)

Figure 7.30: Sample denoised image using the Legacy scheme for Gaussian noise.

| $\sigma$ | image | reference | Legacy (x1) |
|---|---|---:|---:|
| 10 | lena | **35.6** | 34.2 |
| | boats | 33.0 | **34.0** |
| | barb | **34.0** | 32.4 |
| 20 | lena | **32.7** | 31.0 |
| | boats | 29.9 | **30.5** |
| | barb | **30.3** | 28.6 |
| 25 | lena | **31.7** | 30.0 |
| | boats | **30.8** | 29.3 |
| | barb | **29.6** | 27.3 |

Table 7.14: Tabulated legacy results for Gaussian noise. Note that there is no available information for the denoising performance of the SMG algorithm for Boats when $\sigma = 10$.

As can be seen, the results for this setting do not reach the best available objective performances attained by the other algorithms. On the positive side, they are better than a "standard" Wiener filter.

## 7.4.3 Selection of the parameters

The results given in this section will focus on the case $\sigma = 20$ and the images Boats and Lena.

**number of conditioning classes** Figure 7.31 shows the effect of this parameter for the case $\sigma = 20$. This behavior is repeated for the other values of $\sigma$.

**LBG iterations** 7.32 shows the effect of this parameter for the case $\sigma = 20$. Again, this behavior is repeated for the other values of $\sigma$.

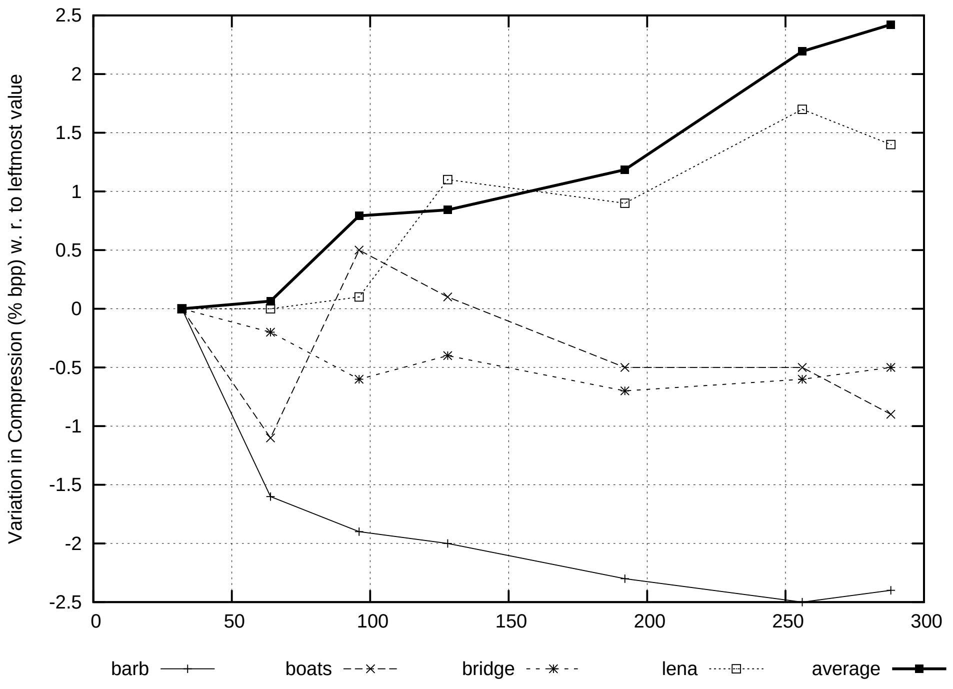
**prediction scheme** The results of 7.15 show the effect of this parameter when the prediction scheme is the average filter. At a late stage of this work, a Gaussian lowpass filter was added which increased the overall performance.

| image | average | median | gaussian |
|---|---:|---:|---:|
| lena | **30.6** | 30.4 | 30.6 |
| boats | 30.0 | 29.8 | **30.2** |
| barb | 28.0 | 27.9 | **28.3** |

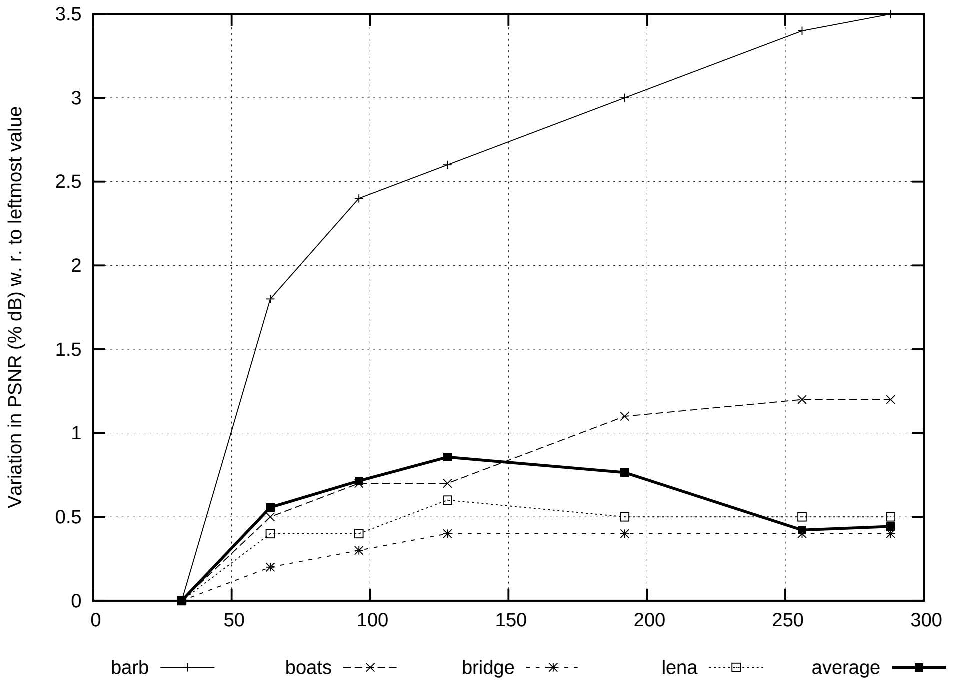
Table 7.15: Results for different prediction schemes.

**recursive prefiltering** Figure 7.33 reveals that the performance actually decreases after one iteration.

Figures 7.34 through 7.36, which show how Lena, Boats and Bar look when denoised recursively, give a hint of what could be the problem. Although a more in-depth analysis is required, one possible explanation is that an unstable *closed-loop behavior* is affecting the recursive application of the DUDE.

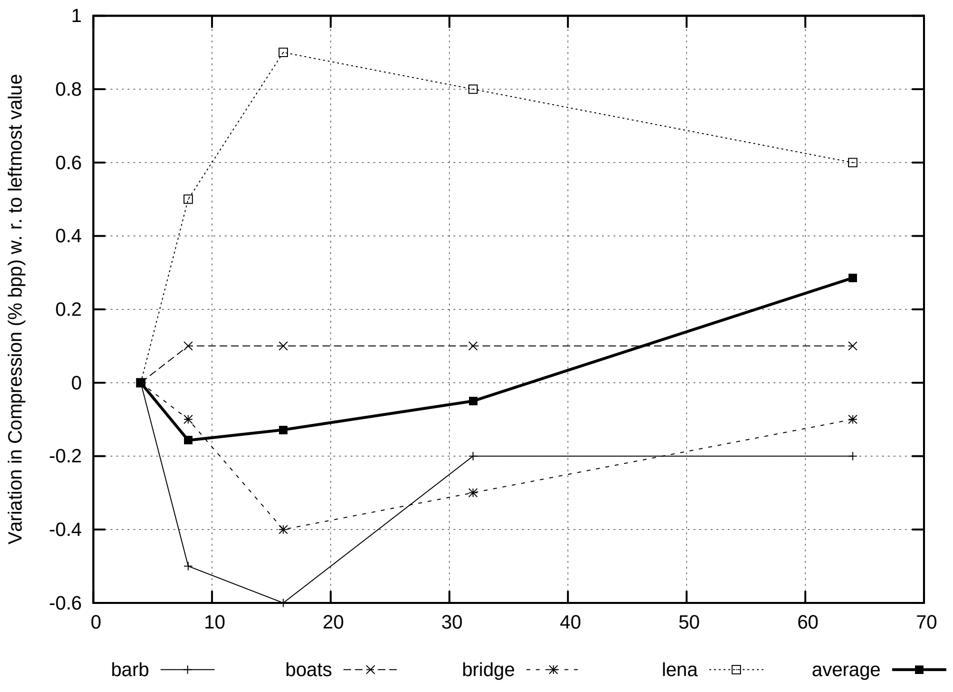


(a) Compressibility.

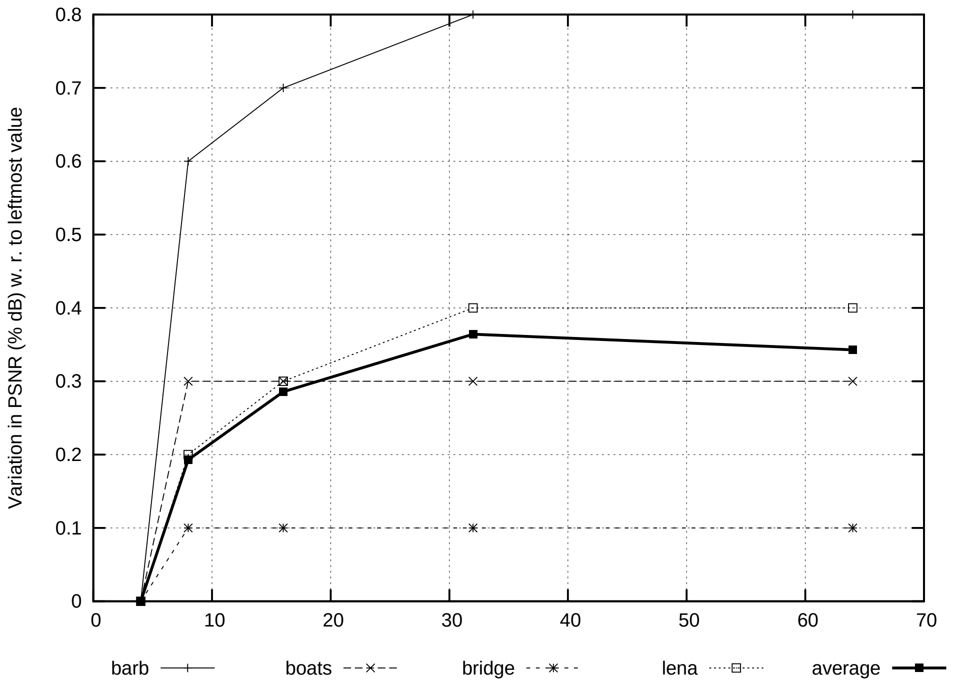


(b) Fidelity.

Figure 7.31: Legacy performace vs. number of context clusters. The value which attains the minimum is near 128 clusters.

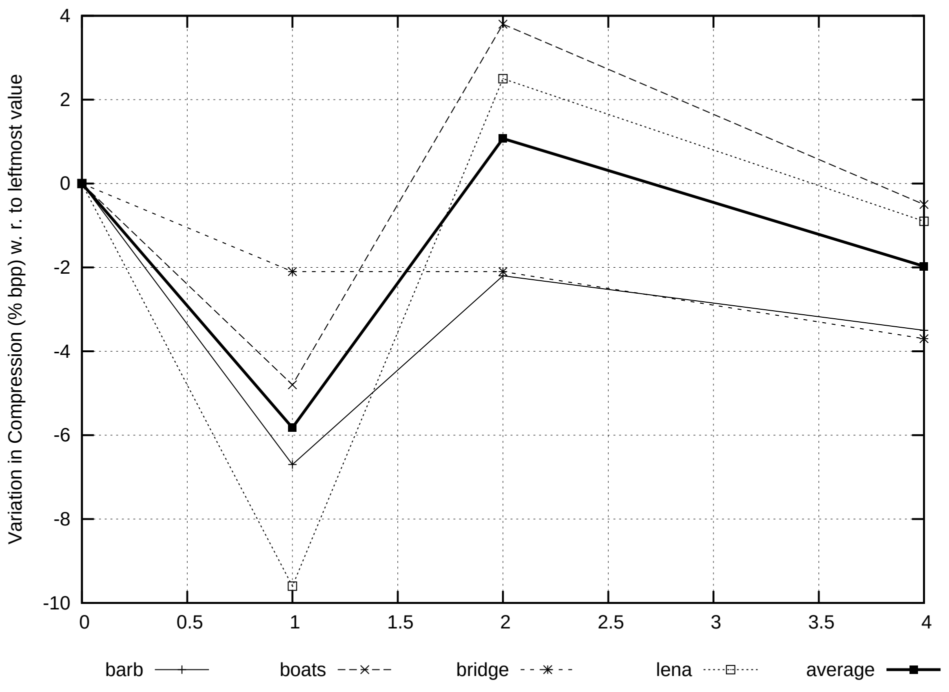


(a) Compressibility.

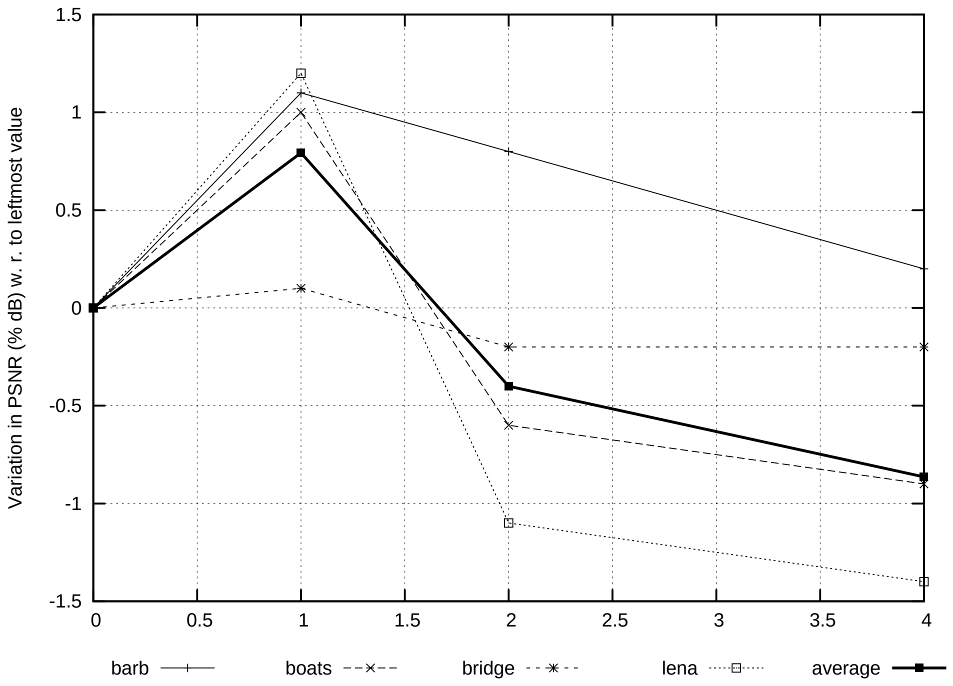


(b) Fidelity.

Figure 7.32: Legacy performace vs. LBG iterations.

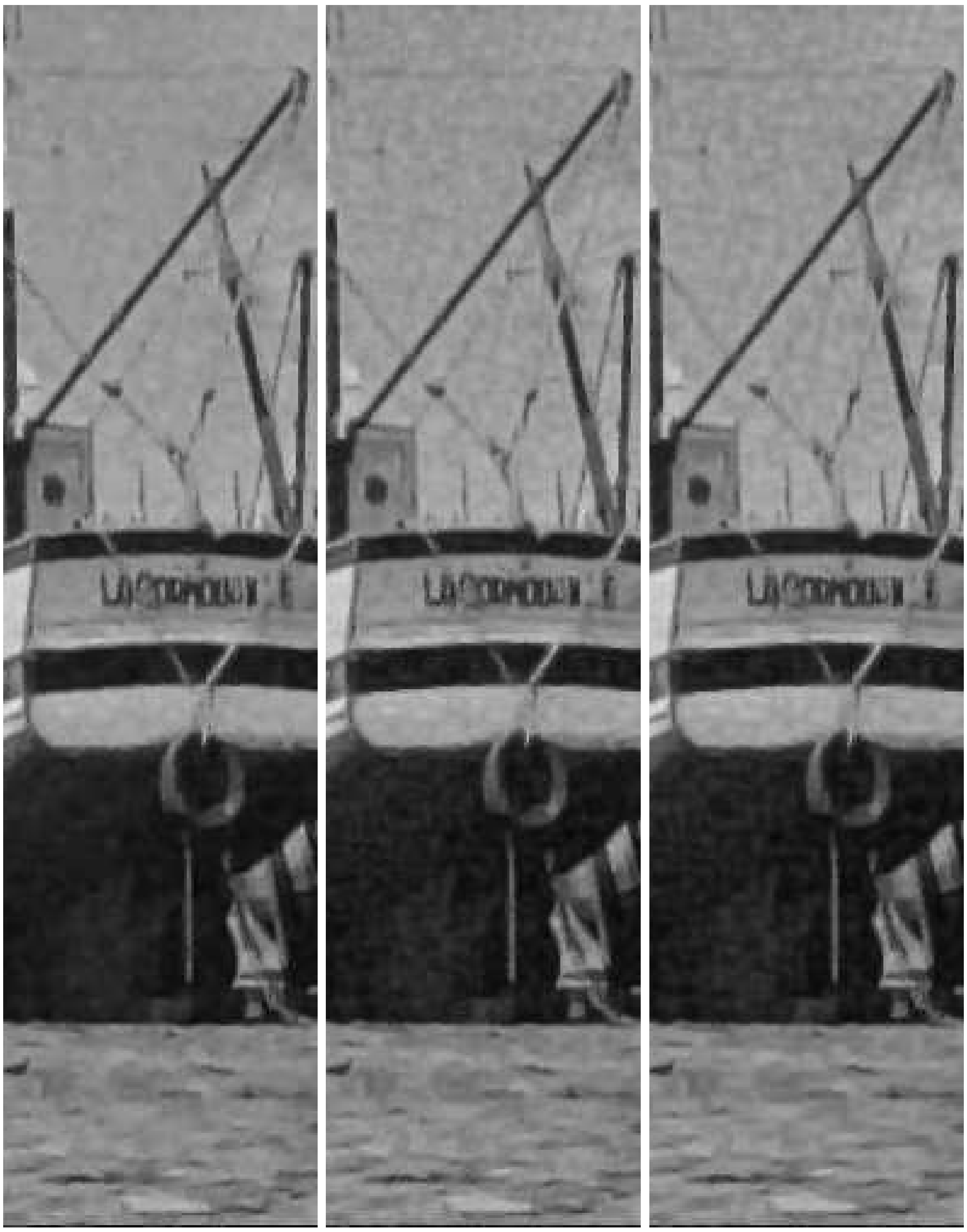


(a) Compressibility.



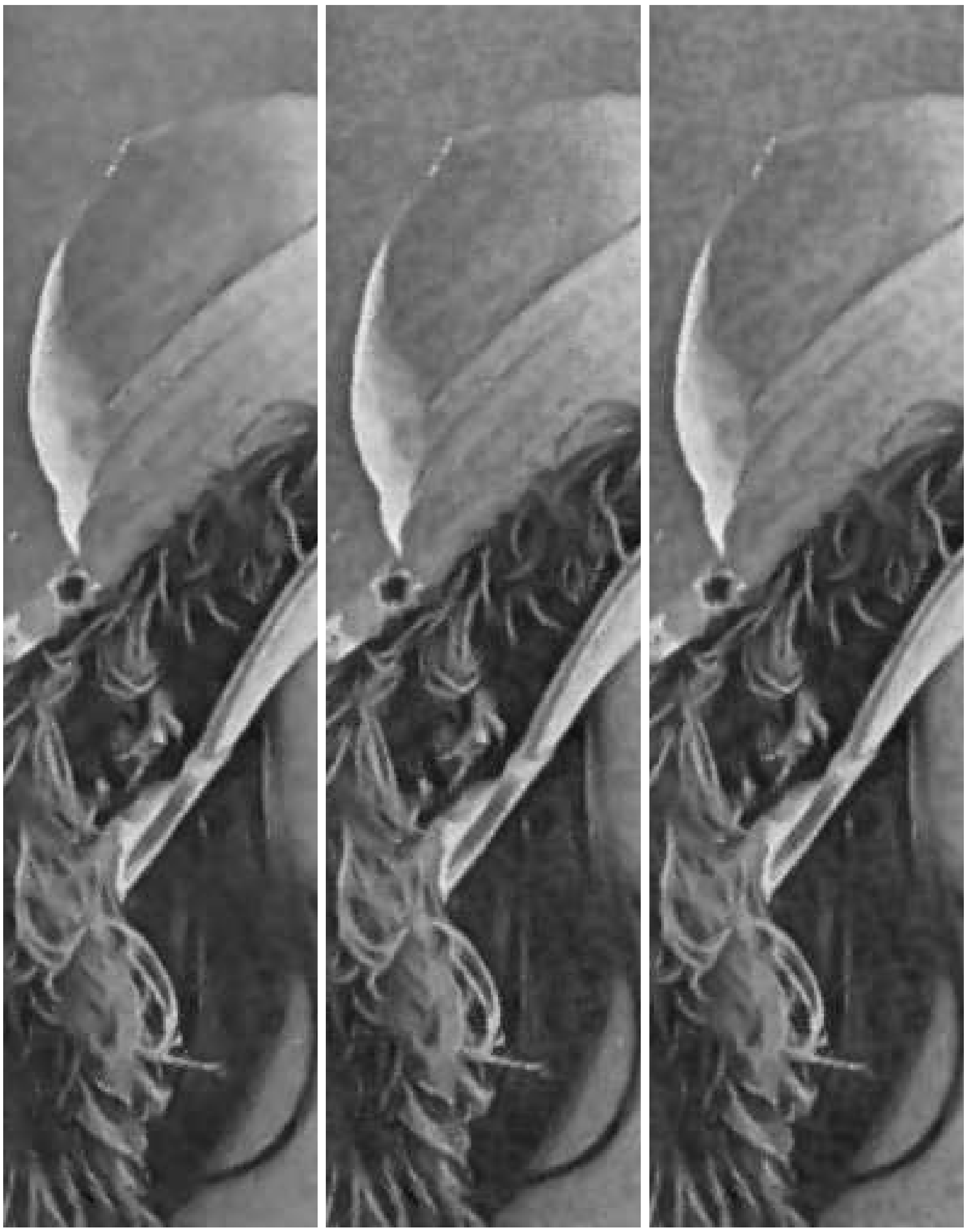
(b) Fidelity.

Figure 7.33: Legacy performace vs. number of recursive prefiltering applications.



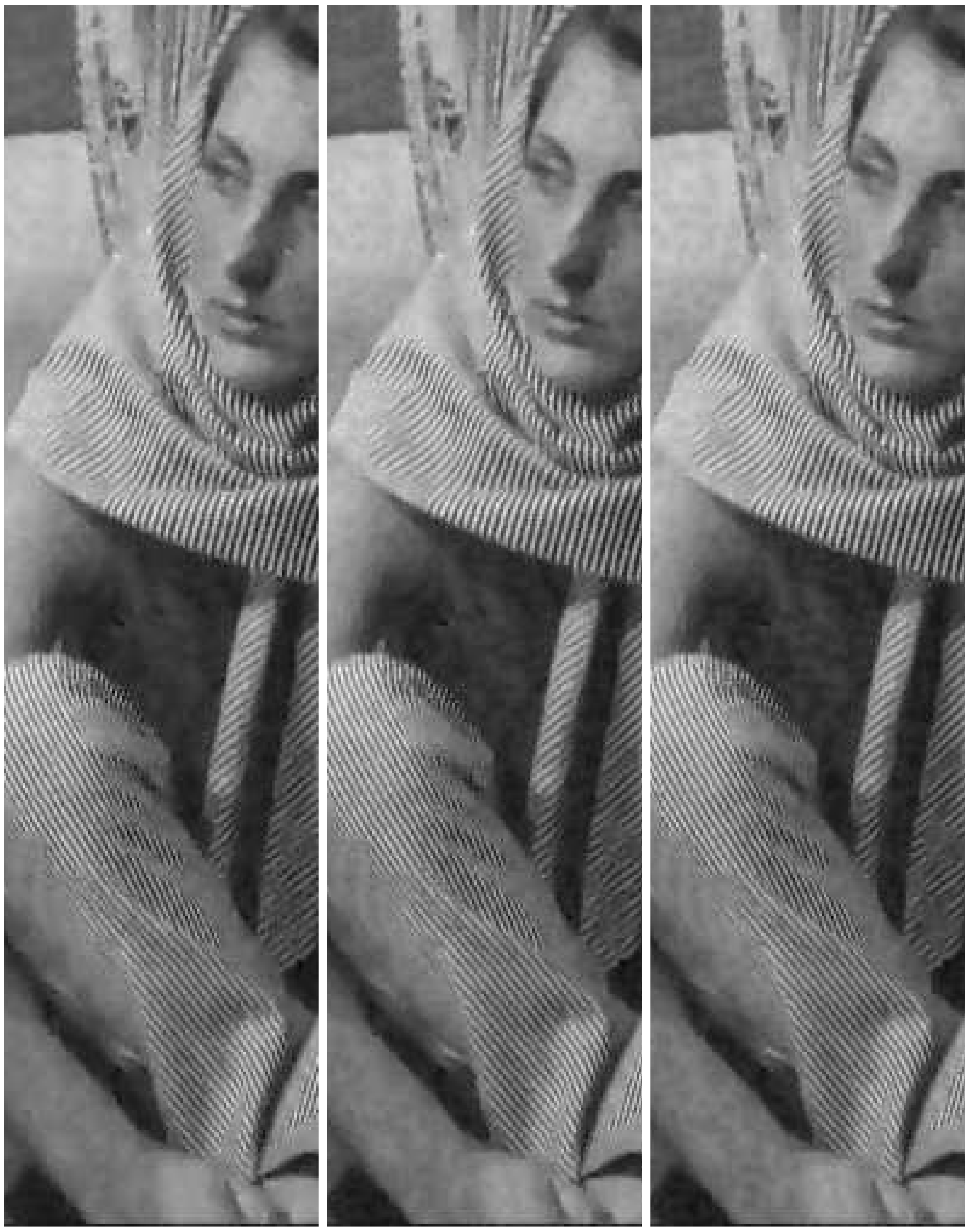
(a) Boats (x1) (b) Boats (x2) (c) Boats (x4)

Figure 7.34: Recursive denoising for the gaussian channel and its effect: Boats.



(a) Lena (x1) (b) Lena (x2) (c) Lena (x4)

Figure 7.35: Recursive denoising for the gaussian channel and its effect: Lena.



(a) Barb (x1) (b) Barb (x2) (c) Barb (x4)

Figure 7.36: Recursive denoising for the gaussian channel and its effect: Barb.

In other words, because each denoised sample $\hat{\mathbf{z}}_i$ depends on the prediction,

$$\hat{\mathbf{x}}_i = g(\mathbf{z}_i, \gamma_i, \hat{\mathbf{z}}_i)$$

then the denoised image is a function of the prediction

$$\hat{\mathbf{x}}^{m\times n} = f(\mathbf{z}^{m\times n}, \hat{\mathbf{z}}^{m\times n})$$

On the other hand, the prediction $\hat{\mathbf{z}}^{m\times n}$ is a function of the prefiltered image $\mathbf{y}^{m\times n}$,

$$\hat{\mathbf{z}}^{m\times n} = \phi(\mathbf{y}^{m\times n})$$

If the output of the denoiser in iteration $n-1$ is used as the prefiltered image of iteration $n$, $\mathbf{y}_n^{m\times n} = \hat{\mathbf{x}}_{n-1}^{m\times n}$ ,and the result is a nonlinear recursive equation on the sequence of denoised images $\hat{\mathbf{x}}_n^{m\times n}$ (here the subindex indicates recursion level),

$$\hat{\mathbf{x}}_n^{m\times n} = f'(\mathbf{z}^{m\times n}, \hat{\mathbf{x}}_{n-1}^{m\times n})$$

where the prefiltering in the first iteration ($n = 0$) is done by some non-recursive filtering function $g(\cdot)$),

$$\mathbf{y}_0^{m\times n} = g(\mathbf{z}^{m\times n})$$

.

This closed loop can be seen in Figure 6.2. If the prefiltering or prediction functions produce similar side effects in each recursion, the effect can become more and more noticeable.

| $\sigma$ | avg/12 | avg/24 | nap1 | nap3/12 | nap3/24 |
|---|---|---|---|---|---|
| 5 | 37.8 | 37.9 | 38.0 | 38.0 | 37.9 |
| 10 | 33.9 | 34.1 | 33.8 | 33.9 | 33.9 |
| 20 | 30.1 | 30.2 | 29.4 | 29.4 | 29.9 |

Table 7.16: Results for different prediction schemes for different $\sigma$ values.

### 7.4.4 Napkin

Although the DUDE-I was designed with all the channel models described earlier in mind, the experimentation was mainly focused on tuning the DUDE-I to the non-additive noise channels for which the DUDE-I yielded outstanding results when compared to the state-of-the-art. The experiments on the Gaussian channel, and especially the application of the Napkin Scheme to it, were performed at the final stages of the present work and are to be considered preliminary.

The initial results for the Napkin modeling approach applied to a Gaussian channel were not as good than those obtained with the Legacy Scheme described earlier when the noise is above certain threshold.

The following experiments in this section are designed to pinpoint the main responsible for this degradation, i.e., either the context modeling component or the prediction component.

**Fixed classification scheme and different prediction schemes** Table 7.16 shows different results in PSNR terms for the Boats image under three different channel parameter values. These results were obtained by fixing the context model using LBG and 256 clusters, and then varying the prediction scheme among five possibilities: average-of-12 (avg/12), average-of-14 (avg/14), Average Napkin Variant (nap1), Smooth Napkin (nap3) (both described in Section 6.6.5) using either the same 12 (nap3/12) or 24 (nap/24) samples.

As can be seen, under low noise levels (less than $\sigma = 10$), the Smooth Napkin does a reasonable work as a predictor. For $\sigma = 20$ the results fall below any of the two average predictors. This indicates that the Smooth Napkin prediction, whose design was aimed at this type of noise, needs further development and experimentation in order to perform well under high noise conditions.

**Fixed prediction scheme and different classification scheme** Table 7.17 shows how the performance varies with the context classification scheme when the prediction scheme is fixed (in this case, to an average of 24). The first is the LBG with 256 clusters (the same used in the previous table), and following it: 16 activity levels (AL/16), 256 activity levels (AL/256) and 256 conditioning classes out of four 2-bit quantized wing gradients (WG/256), and the Broad Variant described in Section 6.6.3 (Broad). The latter was the last of the experiments performed for the Napkin Scheme, and uses 16 Activity Level bits as in the AL/16 case.

The results of Table 7.17 indicate that the Broad Variant is the best configuration for the Napkin modeling scheme in order to perform nearly as well as the LBG scheme . Note that only for the case $\sigma = 25$ the difference between the LBG and the Broad Napkin surpasses 0.5 dB.

If the Napkin context modeling is used, the DUDE-I can be applied to large images. Thus, we finish the discussion with a sample result for the Napkin/Broad variant on Bike (using a window average as the predictor) . The result is shown in Figures 7.37 and 7.38.

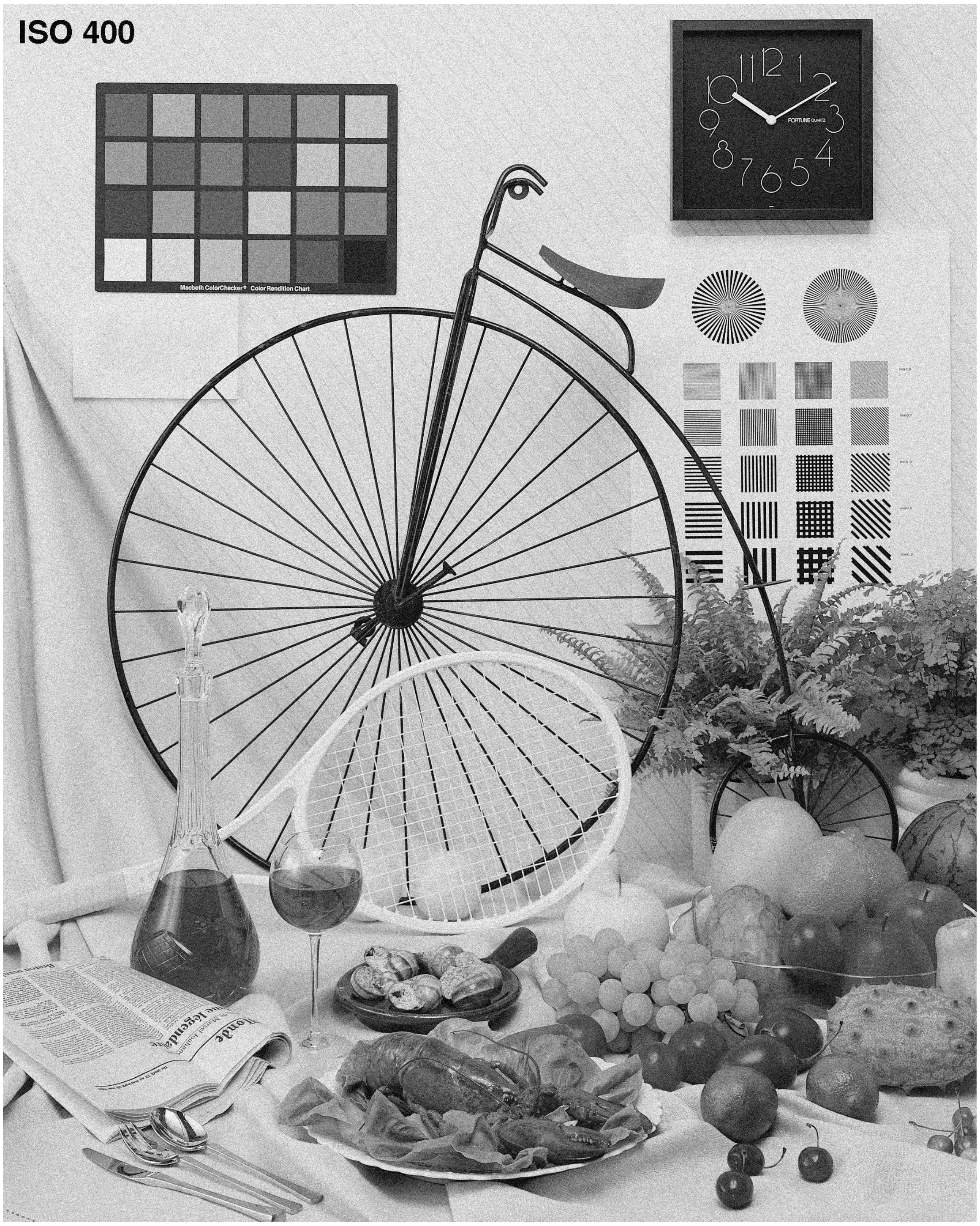


Figure 7.37: Bike corrupted by Gaussian noise with $\sigma = 20$.

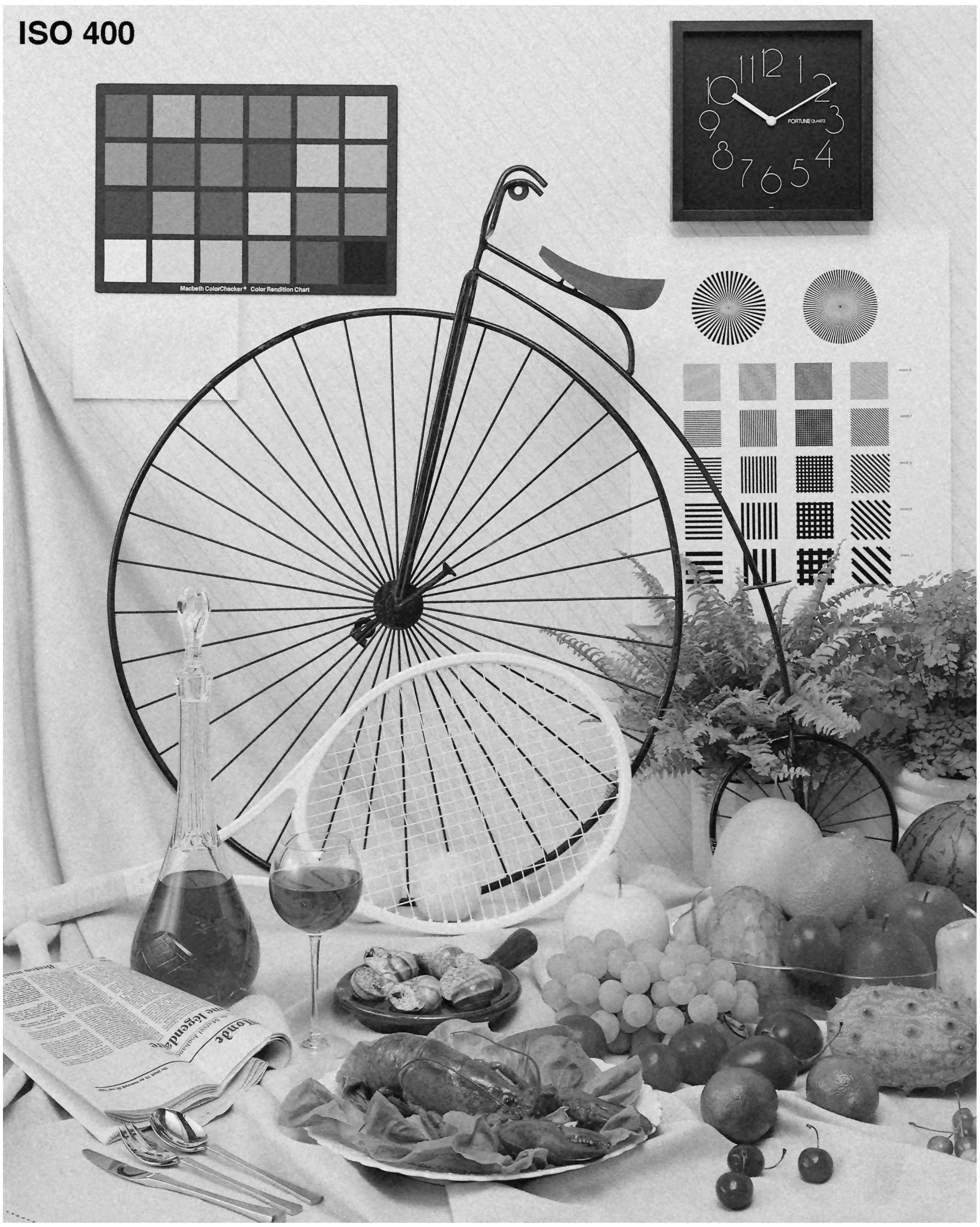


Figure 7.38: Bike denoised using the Napkin/Broad variant as the context classification scheme and a window average as the predictor.

| $\sigma$ | LBG | AL/16 | AL/256 | WG/256 | Broad |
|---|---|---|---|---|---|
| 5 | 37.8 | 37.00 | 36.9 | 36.6 | 37.5 |
| 10 | 33.7 | 32.5 | 32.5 | 32.2 | 33.5 |
| 20 | 30.1 | 28.8 | 28.7 | 28.7 | 29.6 |
| 25 | 29.1 | 27.8 | 27.7 | 27.8 | 28.4 |

Table 7.17: Results for different modeling schemes for different $\sigma$ values.

| $\sigma$ | image | parametric | greedy |
|---|---|---|---|
| 10 | lena | 33.9 | 34.2 |
| | boats | 33.5 | 34.0 |
| | barb | 31.8 | 32.4 |
| 20 | lena | 30.9 | 31.0 |
| | boats | 30.0 | 30.5 |
| | barb | 28.0 | 28.6 |
| 25 | lena | 29.8 | 30.0 |
| | boats | 29.1 | 29.3 |
| | barb | 26.8 | 27.3 |

Table 7.18: Comparative results for the parametric and greedy channel inversion algorithms.

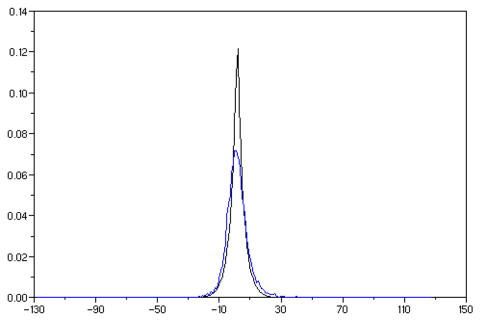
### 7.4.5 Other results

#### Input distribution parametrization

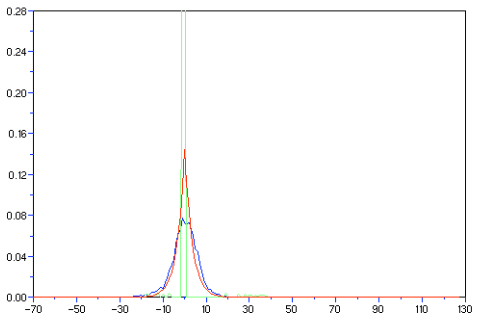
One of the goals for the Gaussian channel was to remedy the ill conditioning of the channel transition matrix for this case. By assuming that the input distribution is a parametric distribution rather than any distribution over the 8-bit inpt alphabet, its parameters are directly obtained from the statistical moments of the output distribution. This eliminates the instability of the solution and, furthermore, the computational cost of a full $256 \times 256$ matrix inversion or the greedy algorithm (which is a costly operation), at the cost of imposing heavy constrains on the input distribution.

Figure 7.39 shows the comparison of two sample **real** conditional input distributions (computed using the noisy context classification as the conditioning classes but over the clean image) and their respective greedy and parametric approximations. In this case, the parametric approximation is very close to the clean distribution. Figures 7.39(c) and 7.39(d) show two cases where the parametric distribution is not as good.

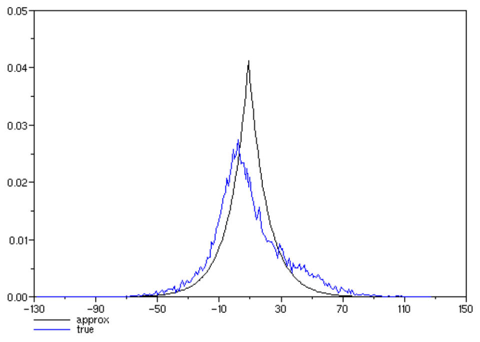
Finally, Table 7.18 shows some sample results using the two approaches. Even tough the input distribution parametric estimations seem to be better than the greedy ones, the results are clearly and consistently better for the greedy algorithm. As the greedy clearly favours the center value, it could be argued that, as in the impulse case, the prediction is the main responsible for the results. However, the results of using the **true** distributions for denoising (*cheating* the channel inversion process) are better than those of the greedy. One possibility is that the cases of 7.39(c) and 7.39(d) is actually so bad that it drops the overall performance even tough the estimation is good in other cases, but this is still speculation and no sound conclusion is available at this time.



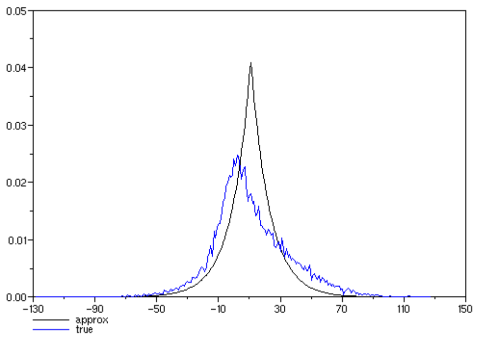
(a)



(b)



(c)



(d)

Figure 7.39: a) Sample true input distribution and its parametric estimation. b) The high peak is the greedy approximation, the other two are the real distribution and its parametric estimation (smoother). c) and d) are two cases where the parametric estimation is not so good. This example was produced with the Baboon image.

### 7.4.6 Sensibility to the channel parameter

Figures 7.40 (for $\sigma = 10$) and 7.41 (for $\sigma = 20$) show how the denoising performance is affected when the true channel parameter is $\sigma$ and the denoising is performed using an estimated parameter $\sigma'$.

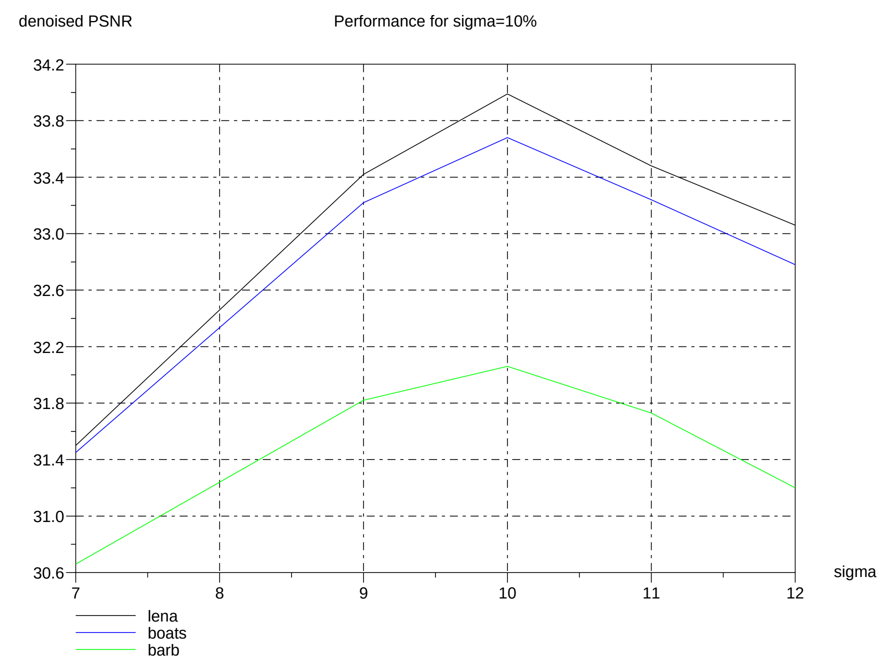
As expected, the performance is decreased as the difference between $\sigma$ and $\sigma'$ increases. In any case, the performance is never below the performance of a Wiener filter for an error in the parameter $\sigma'$ of 15%, and is above the image noisy PSNR (i.e., it still performs some denoising) even for an error of 30% (see Table 7.13).

### 7.4.7 Sensibility to pseudo-random noise generation

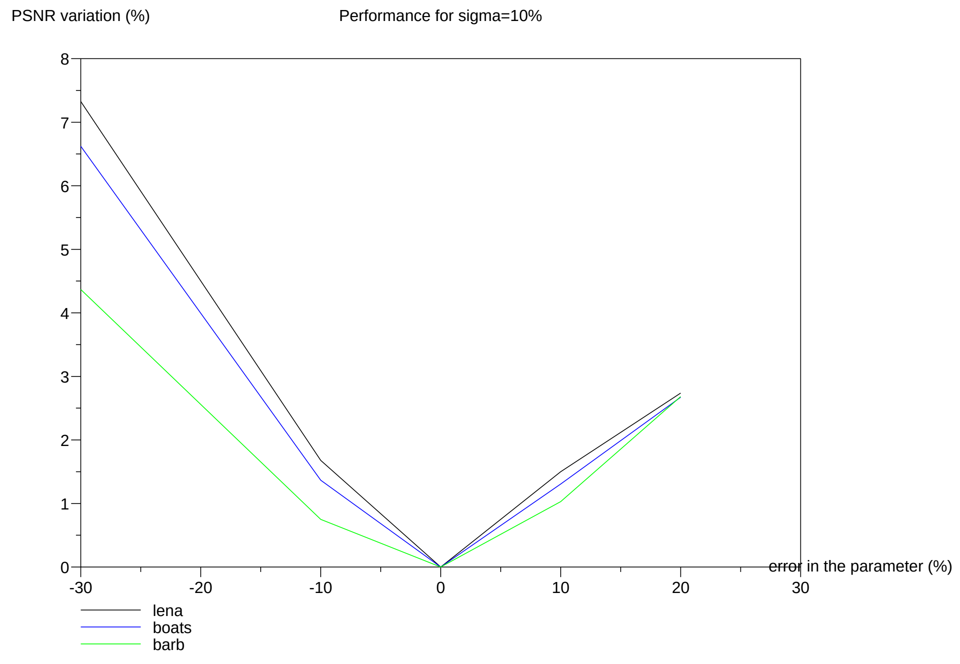
As for the impulse noise, the dependency of the results with respect to the random seed of the random noise generation function used to produce the noise is studied. Table 7.19 shows the results.

| image | min | max | mean | std. dev. |
|---|---|---|---|---|
| lena | 30.62 | 30.69 | 30.66 | 0.02 |
| boats | 30.04 | 30.17 | 30.12 | 0.04 |
| barb | 28.01 | 28.11 | 28.05 | 0.03 |

Table 7.19: Sensibility of the result for 8 different random noise simmulations. These results are obtained using the Legacy modeling scheme with no recursive application.

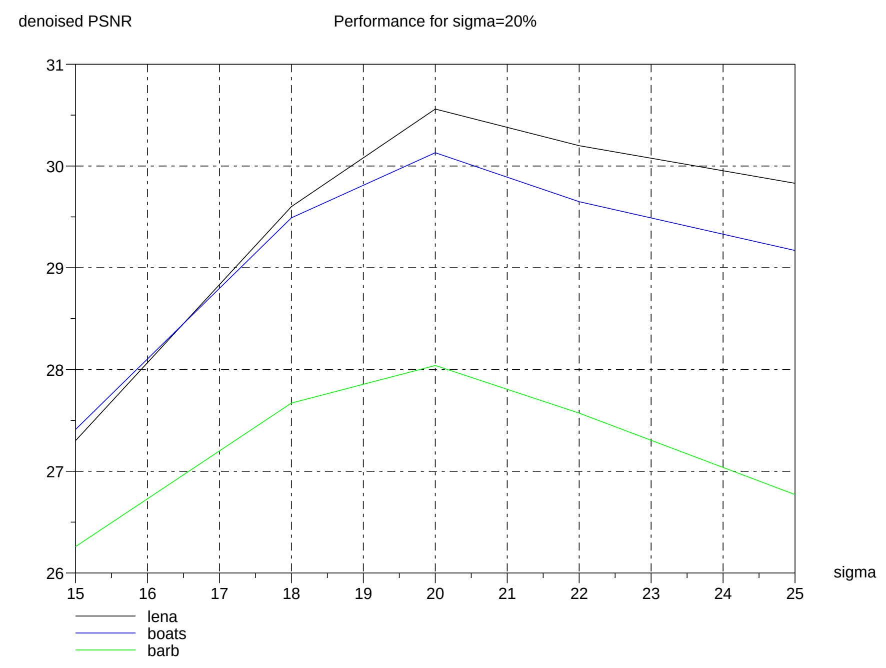


(a)

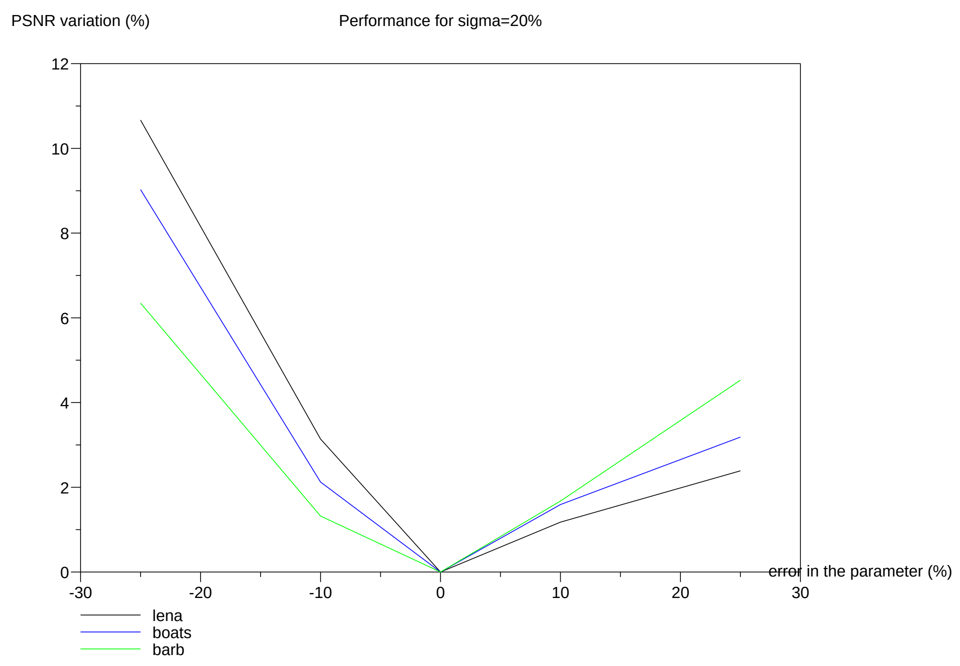


(b)

Figure 7.40: Sensibility of the denoiser for the case $\sigma = 10$. a) absolute value, b) decrease in performance proportional with respect to the case $\sigma' = \sigma$.



(a)



(b)

Figure 7.41: Sensibility of the denoiser for the case $\sigma = 20$. a) absolute value, b) decrease in performance proportional with respect to the case $\sigma' = \sigma$.

## 7.5 Real life denoising

The discussion so far has been centered on the application of the DUDE-I algorithm to simulated noisy images where all of the properties (especially memoryless nature) on the channel and its parameters are perfectly known. As a practical application, the ultimate goal is to apply it to images corrupted by real noise. This section presents some preliminary results for the DUDE-I when applied to the image shown in Figure 7.42. This is a scanned page of an ancient translation of the work of Euclides to the Spanish language where the ink from the reverse page has filtered thru to the front page. Clearly, this is not memoryless noise as the noisy samples mantain the rough shape of the reverse page letters. However, this structure is revealed at a higher scale, a fact that could be exploited by the DUDE-I framework by restricting the memoryless attribute of the channel to be "local". Even under this assumption, there are two more questions to answer: which channel model to use? which paremeters? As there is no "real" noise channel here, a channel has to be selected which performs best and thus the channel and its parameters become parameters of the algorithm rather than part of the problem specification.

Figure 7.43 shows one of the best results obtained for this image. As there is no clean version of the image, a hand cleaned version of this image is used as a reference. Figure 7.44 compares this result with other methods. The parameters for this result are:

**Channel** Gaussian with $\sigma = 20$.

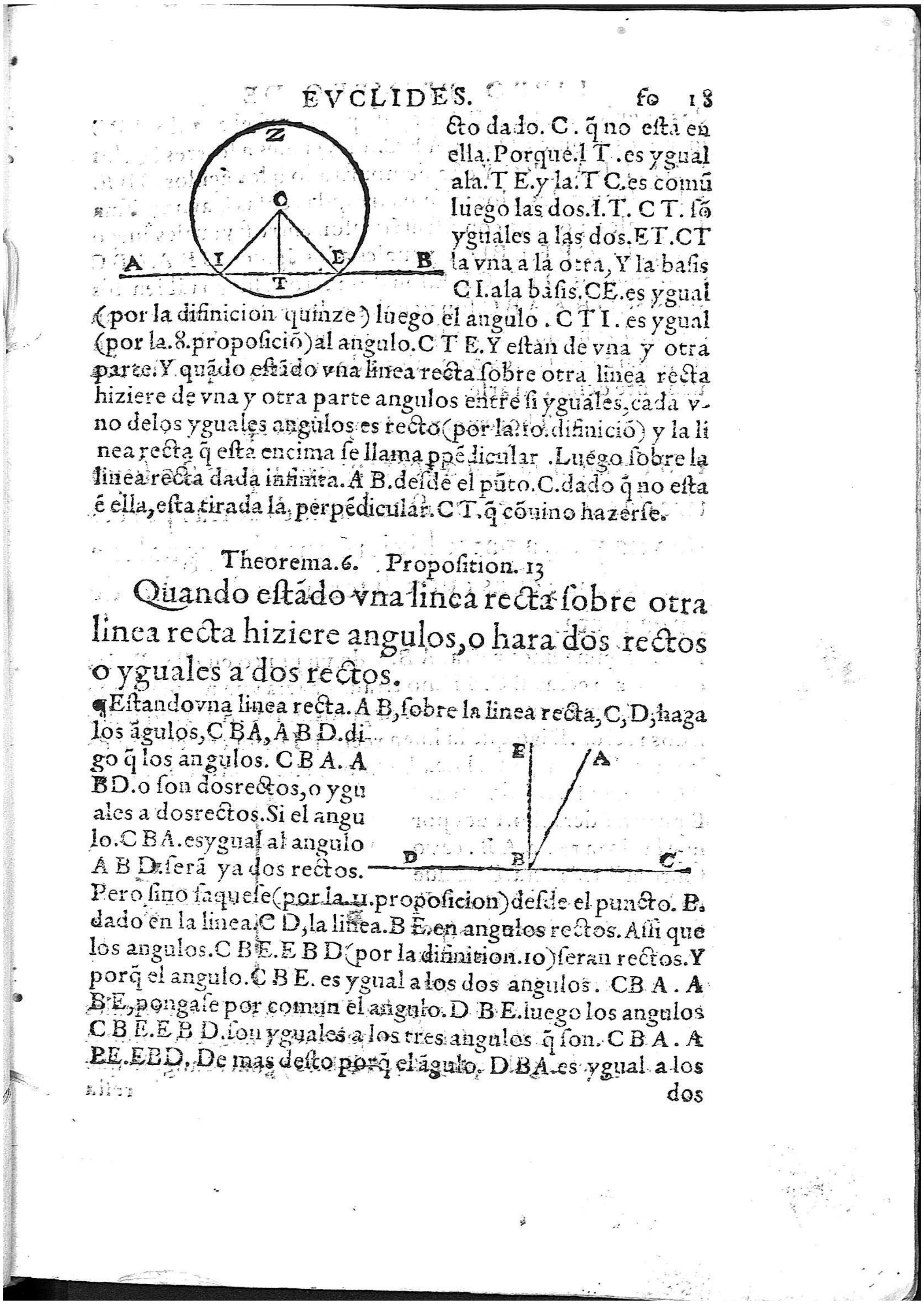
**Prefilter** Median of $3 \times 3$ square window.

**Modeler** Legacy modeling with the following configuration:

- $5 \times 5$ square neighborhood contexts.
- Only 8 context classes.
- Average of $7 \times 7$ square neighborhoods filter as a predictor.

As can be seen, the denoised version is closer to the hand cleaned version than the original one than the output of any of the other algorithms. Another interesting point is that the parameters are quite different to the ones that have been used to denoise "real gaussian" noise so far. These were obtained after many experiments on the different parameters in the proximity of certain initial guesses.

**The DUDE-I as an interactive denoising tool** The results in this section are clearly appart from the rest of the discussion and must be considered only as a first hint on the utility of the DUDE-I framework as a semi-automatic denoising tool for real life image denoising. In this case, the channel type and parameters are additional user selectable aspects of the algorithm that could, for example, be chosen interactively in appropiate dialog boxes (such as in the Adobe PhotoShop or GIMP filters).



EVCLIDES. fo 18

cto dado. C. q̃ no eſta en ella. Porque. I T. es ygual ala. T E. y la. T C. es comũ luego las dos. I. T. C T. ſõ ygủales a las dos. E T. C T la vna a la otra, Y la baſis C I. ala baſis. C E. es ygual (por la difinicion quinze) luego el angulo. C T I. es ygual (por la. 8. propoſiciõ) al angulo. C T E. Y eſtan de vna y otra parte. Y quãdo eſtãdo vna linea recta ſobre otra linea recta hiziere de vna y otra parte angulos entre ſi yguales, cada vno delos yguales angulos es recto (por la. 10. difiniciõ) y la linea recta q̃ eſta encima ſe llama ppẽdicular. Luego ſobre la linea recta dada infinita. A B. deſde el pũto. C. dado q̃ no eſta ẽ ella, eſta tirada la perpẽdicular. C T, q̃ cõuino hazerſe.

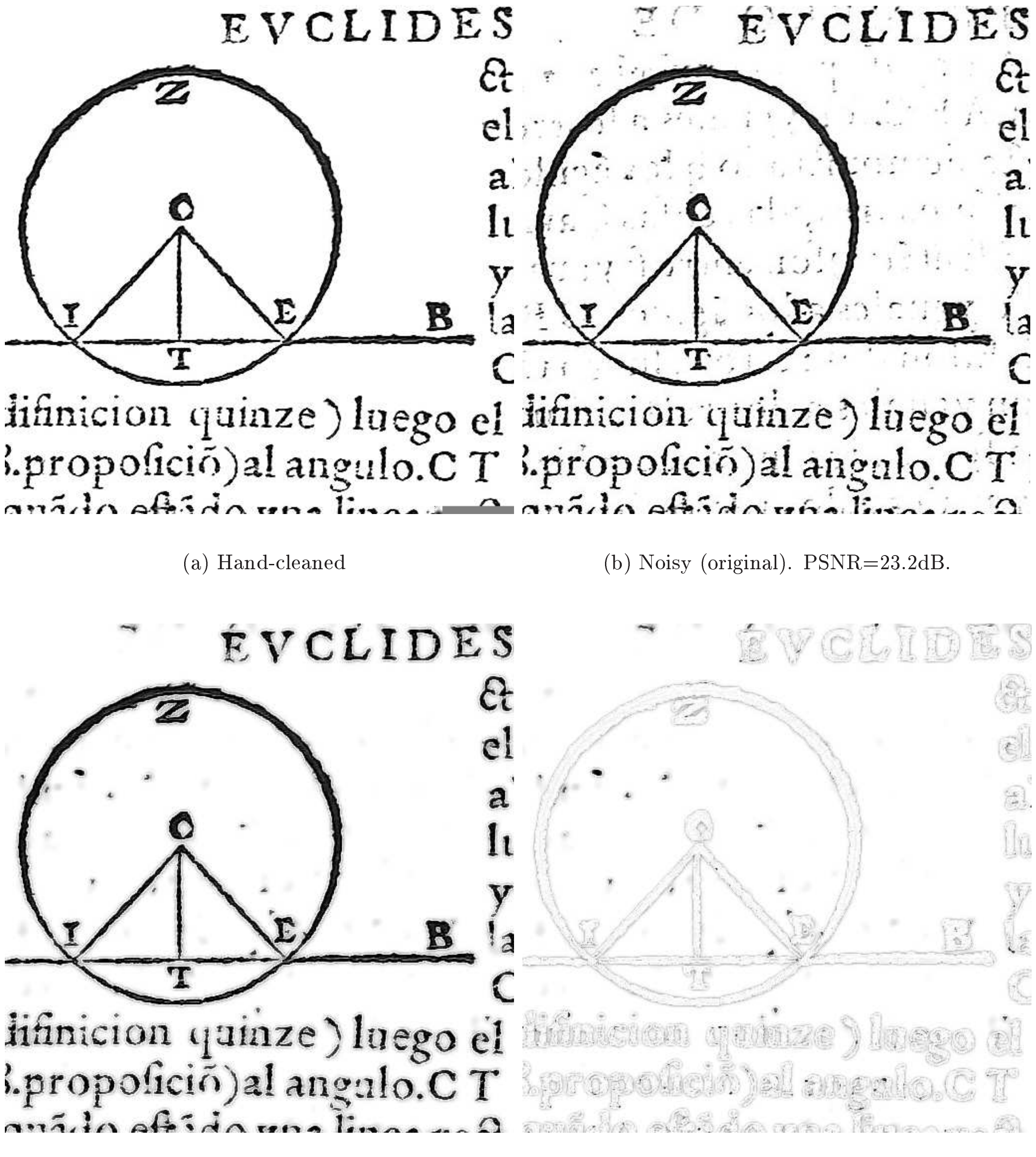
Theorema. 6. Propoſition. 13

Quando eſtãdo vna linea recta ſobre otra linea recta hiziere angulos, o hara dos rectos o yguales a dos rectos.

¶ Eſtando vna linea recta. A B, ſobre la linea recta, C, D, haga los ãgulos, C B A, A B D. digo q̃ los angulos. C B A. A B D. o ſon dos rectos, o yguales a dos rectos. Si el angulo. C B A. es ygual al angulo A B D. ſerã ya dos rectos. Pero ſino ſaqueſe (por la. 11. propoſicion) deſde el puncto. B. dado en la linea. C D, la linea. B E. en angulos rectos. Aſſi que los angulos. C B E. E B D (por la difinition. 10) ſeran rectos. Y porq̃ el angulo. C B E. es ygual a los dos angulos. C B A. A B E, pongaſe por comun el angulo. D B E. luego los angulos C B E. E B D. ſon yguales a los tres angulos q̃ ſon. C B A. A B E. E B D. De mas deſto porq̃ el ãgulo. D B A. es ygual a los

dos

Figure 7.42: Scanned Euclides page (page034)


EVCLIDES
Z
O
I
E
B
T
T

(a) Hand-cleaned

(b) Noisy (original). PSNR=23.2dB.

(c) DUDE-I. PSNR=23.3dB.

(d) Absolute difference.

Figure 7.43: Best result for the Euclides page. The PSNR is interpreted as the difference measure with to the hand cleaned version.

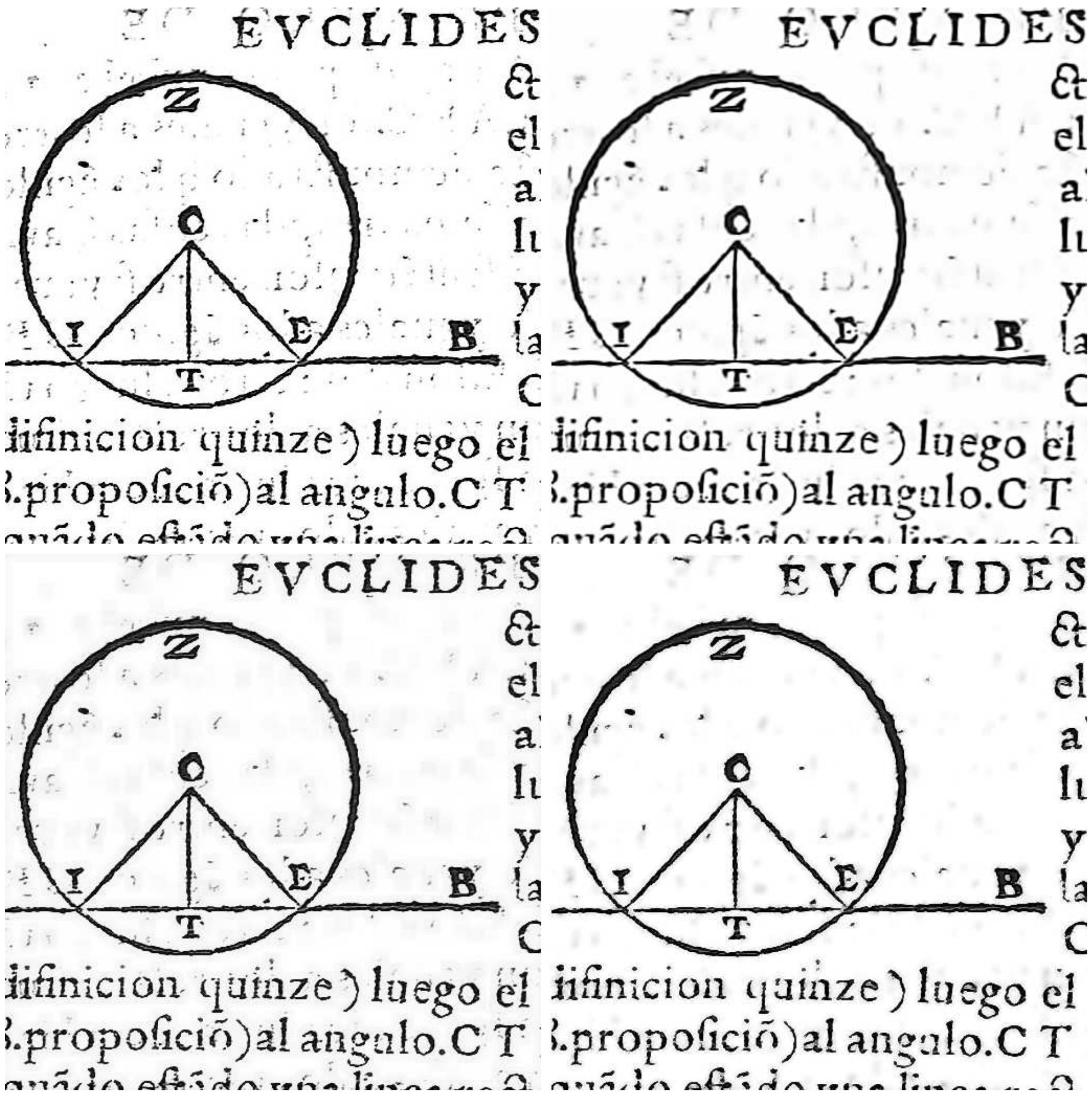

EVCLIDES
Z
O
I
E
B
T
lifinicion quinze) luego el
.propoſiciō)al angulo.C T

Figure 7.44: Clockwise starting with top-left: scanned image; GSM [25]; DUDE-I; Wiener Filter (MatLab `wiener2` function). PSNR relative to hand-cleaned version: 23.2 dB(scanned), 22.6 dB(GSM), 25.9 dB(DUDE-I) and 25.2 dB(Wiener).

# 8 Concluding remarks

## 8.1 Overall

The main goal of this work was to adapt modeling tools used successfully in image compression such as context modeling and prediction to the DUDE algorithm in the hope that, by doing this, it would be possible to address the denoising of continuous tone images using this paradigm.

An augmented framework was defined, and it proved to give good results for various types of noise, surpassing the current state-of-the-art in some cases. Furthermore, this framework can be extended, and better modeling schemes can be built on top of it which could solve its current limitations. Although several problems remain that require further research, significant progress was achieved towards this the goals defined.

Of the channel models that were used to test the new system, the results that were obtained are very good when compared to the state of the art in the case of non-additive noise types. For the additive Gaussian channel , the results are below the best available results, although, at the same time, significantly above the results that can be obtained using a classical Adative Wiener filter.

The results for the impulse channel indicate that the main contribution to the performance is due to the success of the prediction scheme rather than in the context modeling part. However, this is a side effect of the very particular case of the impulse channel where the DUDE-I automatically chooses the correct behavior by letting the predicted value be the main influence in the decision of the output. The added computational burden is small and the resulting framework is more flexible than a hard-coded specific filter for impulse noise removal.

## 8.2 Modeling approaches

At a general level, the Canonical Transformation and DC cancellation tools, which are applied in every modeling algorithm used, have proven to improve the overall performance, which indicates that one of the main aspects to look at when doing a context model is to exploit the potential symmetries that exist in the structure of images.

The first attempt at context modeling was to use the LBG algorithm to do a vector quantization of the contexts. Even tough its use led to some of the best results in terms of denoising performance, this model has shown to be impractical, mainly because its computational requirements are too high even for small images.

The Napkin modeling has shown to be a good scheme for the Impulse noise. In this case, most of the performance gain is credited to the prediction scheme, including the role of the context modeling scheme when applied to the predictor context-dependent bias cancellation. Of the different features that the context model of the Napkin scheme has to discriminate contexts, the texture and activity level components were found to be the most useful. The first is specially useful for bias cancellation, while the second helps to sepparate the flat regions of the image from the borders, resulting in a good adaptive prediction scheme. Despite this, the results in Section 7.2.3 indicate that features such as the gradient direction or wing gradients provide useful information on the structure of the contexts, and thus these features should be subject to a deeper analysis before ruling them out.

This model needs further development to achieve the desired robustness to additive noise, performing about $0.3dB$ below the Legacy results when using the Broad Variant described in Section 6.6. However, this disadvantage could be overweighted by the reduced computational requirements implied by this method, which allow the application of the DUDE-I to large images such as Bike.

## 8.3 Noise types

### 8.3.1 Non-additive noise

As mentioned, the results for the impulse noise are very good. These results also extend to the case of asymmetric impulse noise and the Z-Channel, and also to the more difficult $q$-ary symmetric channel. All those results benefit from the preclassification schemes, where a simple thresholding was used for the impulse noise and its variants, and the more sophisticated homogeneity classification for the $q$-ary symmetric channel. The prefiltering scheme proved to be specially useful, being always a simple median filter (in the first iteration). All of this combined with the recursive prefiltering scheme allowed toe DUDE-I to reach and surpass the state of the art.

It must be noted that any denoising algorithm could be used for the prefiltering stage, possibly rising the overall performance. This includes simple but yet better algorithms such as the Adaptive Median described in Section 3. Furthermore, this observation applies to any type of noise.

Of the two modeling schemes, the LBG gives slightly better results than the Napkin at the cost of being much heavier in terms of computational resources. The only exception among all the images of the test suite (not only the ones shown here) is the Barb image, where the difference is very noticeable between the two approaches. On the average, the best tradeoff between the two, for the impulse noise, is the Napkin scheme.

### 8.3.2 Gaussian noise

The gaussian channel has proven to be more challenging to the proposed scheme, and while the results are not bad when compared to “simple” denoising strategies like a Wiener filter, they are always below the state of the art.

Of the available modeling schemes, the slower LBG yielded the best results. The faster alternative provided by the Napkin context modeling scheme (using the broad version of the gradient wing computations and a context average as the prediction scheme) decreases the overall performance by about 0.5 dB with respect to the LBG approach, but also enables the application of the DUDE-I to large images where the Legacy scheme results impractical.

A simple prediction scheme such as an average of the context samples was seen to be more suitable than the Napkin edge-detection approach, which appeared too sensitive to the additive noise.

Although the prefiltering scheme, whose purpose original purpose was to attack the non-additive noise cases, did not give good results as an initial prefilter when the filter used was a simple median or average, *one* recursive prefiltering application of the DUDE-I *did* increase the performance in all the cases. More prefiltering applications only degraded the performance.

Another difficult aspect of this channel was the computation of the context-conditional input distributions as the transition matrix is ill conditioned even for small values of $\sigma$ and the inversion is not reliable numerically. Of the two alternatives proposed to solve this problem, the greedy algorithm as proposed in the first approach to this problem by Giovanni Motta is the one that gives the current best results. The parametric approach, which can be considered as a preliminary attempt, yields slightly lower results (less than $0.5dB$, with an average of $0.3dB$). On the other hand it is considerably faster and requires less memory.

# 9 Future work

## 9.1 Modeling schemes

### 9.1.1 Napkin enhancements

The Napkin modeling scheme was strongly influenced by the tools and concepts that are used in compression. Specifically, the context modeling scheme was meant to produce a good discrimination in terms of prediction error statistics. However, these tools are not designed to work for noisy images, and the measurements taken in the Napkin model to give it some robustness have not worked as expected. One possibility is thus to continue on this line, trying to achieve the desired robustness while still using tools such as activity level, texture bits, and edge-detecting predictors. The case of non-additive noise does not count since the prefiltering stage produces a reasonably smooth image for these tools to work with.

### 9.1.2 Other classification approaches

Still under the context classification approach, other classification schemes can be investigated which produce better results under noisy environments (again, mostly for additive noise). Frequency domain techniques [9], wavelets [16] are examples of tools that can give useful context information in the presence of noise.

### 9.1.3 All for one, one for all

The probability models defined by both LBG and Napkin perform a *partition* of the context space into disjoint context classes. However, there is no compelling reason for the disjointness of the classes. The extreme case of this paradigm is given in [1], where every context contributes, in an appropriately weighted form, to the denoising of every location in the image. However, this algorithm requires $O(n^2$ operations to denoise an image with $n$ pixels. This approach is also known as the Parzen Window method for distribution estimation [7, pp. 164–173].

An interesting direction of investigation is to obtain a context modeling scheme that leverages the disjointness of the context classes as the two proposed models do, while keeping the complexity of the algorithm below $O(n^2)$ (for example requiring $O(n \log n)$ operations for an image with $n$ pixels).

### 9.1.4 Context codebooks

The vector quantization performed by the LBG algorithm should yield a small set of representative contexts. This could be applied to a large number of (non necessarily noisy) images in an offline fashion and the resulting context "codebook" be used to produce a fast context classification to be used to denoised any new image that may appear. This could also be applied to the distribution estimations as well.

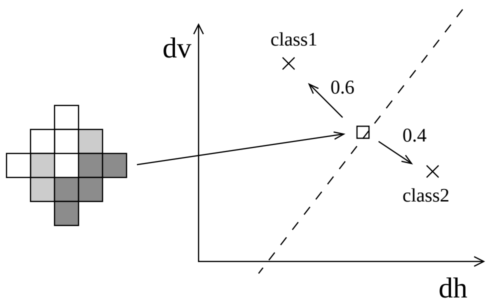


Figure 9.1: Statistics blending concept. Here the context classes are characterized by quantized vectors formed of the vertical and horizontal gradient estimations for the raw contexts. An example context is characterized and its feature vector falls in the middle of two clusters, thus giving a fraction of the "count" to each cluster.

## 9.2 Context statistics

### 9.2.1 Distribution parametrization

It was shown that the use of a simple parametric approach to the problem of gaussian channel inversion is possible. However, the results are still below the ones achieved by the greedy algorithm. Other parametrizations of the empirical distributions should be investigated which give better results. For instance, the proposed two-sided geometric distribution model could be extended to admit nonsymmetric geometric distributions (i.e., where the decay factor $\theta$ is different to each side of the mode of the distribution).

### 9.2.2 Statistics blending

Currently, the probability conditioning model implies a "hard" classification of the raw contexts present in the image into a fixed number of classes. Once this is done, however, the original raw contexts are still available. If the context classes are made up of a certain set of measures (e.g., activity level), and the raw context measures fall at an even distance from more than one context class cluster center in the *measures space*, then assigning the current pixel to one of those classes would incur in a loss of useful information.

Instead of doing this, the contribution of the statistics for each pixel could be divided among several conditioning classes in a way proportional to the likelyhood of the pixel being in each of them (the overall contribution should sum to 1, naturally, as one pixel counts as "1" in the overall statistics). The overall concept is depicted in Figure 9.1.

### 9.2.3 Statistics interpolation

The same idea of 9.2.2 can be used in the second pass when recovering the conditional statistics for the current pixel. If the raw context is recovered (again, for example, an unquantized activity level), a point in the measure space that makes up the context classes can be recomputed. Now, instead of using the nearest class statistics as the conditional statistics for the current pixel (which is what is being done through the conditioning map in the current implementation), one could use a *mixture* of more than one nearest class. Schemes like linear interpolation could be used if the context clusters were produced by scalar quantization on the measure space's dimension. If the quantization is vectorial (e.g., using LBG), then slower but more general algorithms like the *Parzen Window* scheme [7] could be used to produce an interpo-

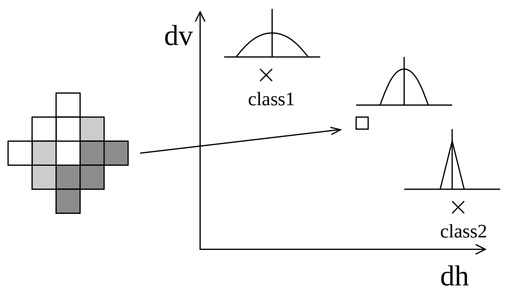


Figure 9.2: Statistics interpolation. Here, each cluster center in the context class feature space has an associated statistics vector to it. By recovering the unquantized feature vector, an interpolated statistics vector can be built out of a number of nearest neighboring clusters.

lated version. Other possibilities are to model the whole image statistics as a multidimiensional field over the conditioning class measure space and apply some *Spline* or polynomial fitting to it. Some of these ideas are depicted in figure 9.2.

### 9.2.4 Tail Bucketing

This is a possible technique to reduce the number of parameters of the overall model. It is based on the idea that the tails of the prediction error density functions would be normally sparse and so the statistics of each symbol on it. On the other side, if the behavior of the predictor (for example, the approximate shape of the prediction error) is known in advance, the sparseness could be reduced by merging all these tails *between* classes and then redistribute the resulting shape among the statistics for each class in lieu of the previous tails. The idea is depicted in Figure 9.3.

## 9.3 Heuristics for noise model type and parameters

A practical issue that needs to be addressed for the DUDE to be used as, for example, a commercial plugin, is to have some sort of noise model and parameter estimation. An ordinary user should not know anything about noise models or parameter, and even a technical user may find it cumbersome to have to specify such parameters each time.

There are many simple techniques for estimating the parameters of channels like the Impulse or the Gaussian channel that could be easily included in a future version.

## 9.4 Automatized parameter selection

This is more a general issue and deals with all the parameters that make up a certain configuration of the DUDE, for example, the size and shape of a context, the number of conditioning states, number of iterations, etc. This has also practical implications if the aim was to obtain a plugin that could be used by non-technical persons.

Many of the current parameters could be automatically chosen once their behavior under different settings (for example, image size) has been studied.

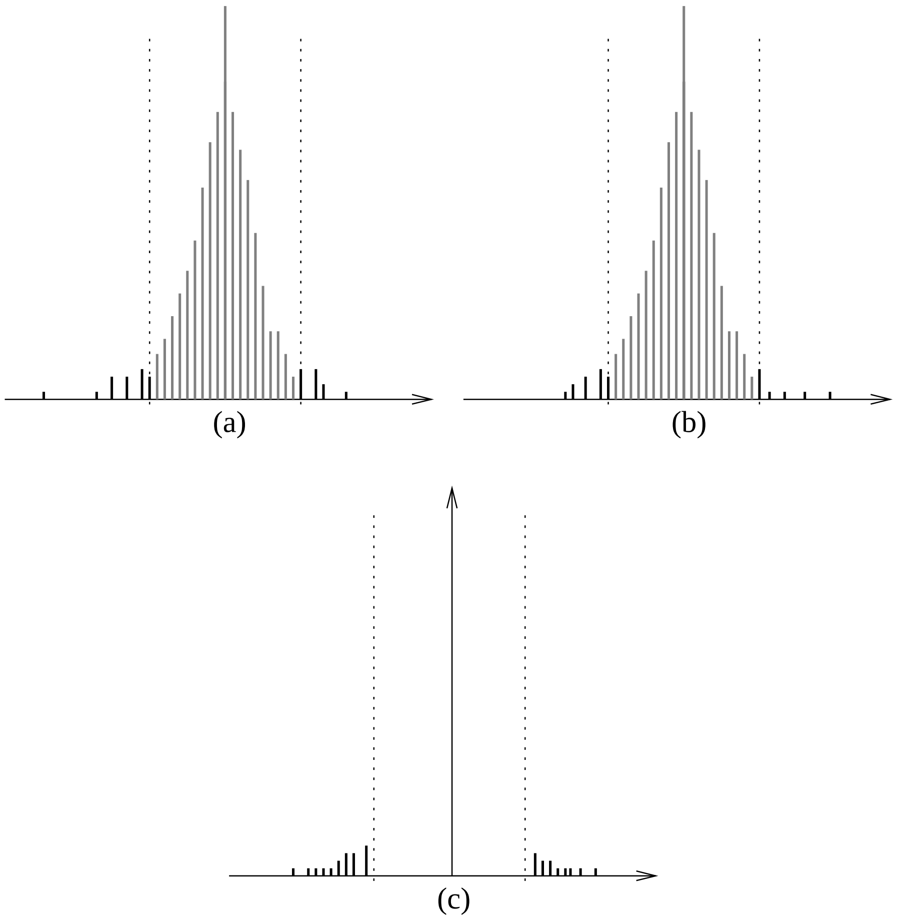


Figure 9.3: Tail bucketing scheme. (a) Prediction error statistics for the two hypothetic context classes. (b) tails are merged into one smoother version, taking scale into account. (c) the resulting shape replaces the original tails.

# A Software implementation

## A.1 Organization

The application was developed in C++ using the GNU C Compiler (GCC) as the main development tool, mantaining cross-compatibility with MS Visual C++ 7.1 (included in MS Visual Studio .NET 2003). For the compilation, both GNU Makefile files and Visual C++ project files are included in the source tree.

**The source code** is extensively documented and conforms to the format used by some autodocumentation tools, specifically, with the Doxygen documentation tool (also available under the GNU Public Licence) which automatically produces a reference manual in various formats including LaTeX and HTML.

**The source code tree** is backed by the Concurrent Versions System (CVS) which is the de facto standard used for version control in most software projects.

## A.2 Source tree

As a general guideline, all the algorithms, including the DUDE implementation itself, were implemented in a modular way without any dependency on the execution environment or user interface. This also applies to the base concept models (sequence, alphabet, channel, etc.) making not only the algorithms but the objects used by them easy to port to other applications.

Almost all the code uses generic programming techniques (C++ templates), as it enables conceptual flexibility while avoiding the overhead related to other common techniques. For example, the `Sequence` class has been generalized to any dimension and symbol type. Furthermore, *template metaprogramming* techniques are used to make any dimension-dependent calculations (for example, D-dimensional indexation) unrolled at compile time.

## A.3 Compatibility

The source code complies with the ANSI C++ standard and currently compiles under GNU Compiler Colection (GCC) 3.x, 4.0 and Visual C++ 7.x. GNU Make makefiles are included for automatic building using GCC, and a Visual C++ Solution is included for VC++ compilation.

## A.4 Version control

The Concurrent Versions System (CVS) version control system was used to manage the project files throughout its development. This is a valuable tool which simplifies the development and update of the project by many programmers and adds redundancy that prevents the loss of data.

## A.5 Prototyping

The Scilab package[1] was used to perform simulations and to analize the results of the experiments. This is a high quality and performance free software alternative to other common simulation environments.

## A.6 Design

The implementation is written in C++ and makes heavy use of *generic programming* concepts (i.e., C++ templates and related techniques) to maximize flexibility and speed at the same time. The design is driven by the Object Oriented Programming paradigm, breaking the problem in a few conceptual families (data, processing blocks, algorithms, utilities) with specific classes representing entities such as *Image*, *Index*, *Context*, *Algorithm*, *Filter*. Each concept family is englobed in a respective *C++ namespace* to clarify the relationship among its members.

**Algorithms** are the central part of the implementation, and are usually broken up into sub-algorithms or *strategies* that can be configured at run time to change specific aspects in the behavior of the algorithm they are part of. They belong to the `algo` namespace. Besides the sequence-specific algorithms (such as *Filter*), several generic algorithms are also included in this module. Examples of these are vector comparison criterions, vector quantization, etc.

The data types orbit around the *Sequence* concept, of which *Image* is a convenient specialization for 2D sequences. To ease the development, classes such as *Sequence* contain declarations for their compatible parameterized related classes. For example, for a 2D sequence, *Sequence* defines an *Index* type which is itself parameterized by D=2. All the sequence-related concepts lie in the `seq` module.

The DUDE-I is implemented as a macroscopic algorithm where the key stages (prefiltering, context modeling, prediction) are governed by corresponding strategies, and the denoising stage depends on the type of noise. Because it is the central algorithm, and because its subalgorithms are actually very complex by themselves, the DUDE is contained in a specific module, the `dude` namespace.

## A.7 Documentation

The full source code is well documented and formatted in a way that enables the automatic generation of printable and/or user-friendly documentation through the *Doxygen*[2] open source automatic documentation tool. The documentation is placed under the `doc/api` directory in the source tree and can be regenerated at any time by typing

```
doxygen dude.dox
```

[1]`http://scilabsoft.inria.fr/`
[2]`http://www.doxygen.org/`

with the root of the source tree as the current directory. The documentation is generated in HTML and LaTeX, and placed in `doc/api/htlp` and `doc/api/latex` respectively. The LaTeXtool has to be run in order to produce a printable document. To do this, type

```
latex refman.tex
```

from the `doc/api/latex` directory.

## A.8 parameterization

As was mentioned, most of the core classes are C++ templates. Two parameters were considered in the generalization of the algorithms: the dimension of the sequences, and the type used for the symbols. In this way, the current implementation is potentially applicable to arbitrary dimensional sequences (from audio to multidimensional images), and arbitrary symbol types (from bytes to double values).

However, some minor changes are needed to be able to use the implementation for dimensions other than 2. This is mainly because some 2D-specific algorithms (for example, the Napkin predictor) are defined only in terms of 2D sequences, and because the code includes the generation of some debugging images which rely on 2D-specific output formats. It is very easy to comment out these parts, and the included 2D-specific algorithms so that the rest works for other applications.

## A.9 utilitles

Flexibility, ease of configuration and runtime debugging output were considered of key importance in the development, as this implementation is an experimentation tool above anything else. A set of general purpose utilities were included that deal with such tasks. These utilities are grouped under the `util` namespace and, because they do not rely on generic parameters, can be precompiled into a library whose name is simply "dude" (actually, the system dependent name may vary: for Windows it is dude.lib, and for Unix/linux libdude.a).

### A.9.1 configuration

All the algorithms are configurable in a hierarchical fashion. Each algorithm can have its own parameters, and its subalgorithms as well. The parameters are organized in a hierarchical, domain-like structure that reflects the aggregation of algorithms and subalgorithms. For example, the DUDE algorithm has its parameters in the "root" domain, thus the name of its parameters appear directly as, for example, "recursion_level" or "recursive". The DUDE includes a prefilter as one of its subalgorithms, configured through the parameter "filter". Filter, in turn, has its own parameters, for example, "template". The latter would appear as a global configuration parameter under the name "filter.template", showing that it is a parameter of the subalgorithm "filter". This same scheme can continue to any depth.

The Configuration tools enable us to use a uniform interface to configure the algorithms and publish the available parameters, regardless of the "front-end". For example, the current implementation can read and write unix-like ASCII configuration files, parse command line arguments and produce help messages to the console without the need to write specific code in the command line interface. The same implementation is used by the GUI to configure the underlying implementation.

### A.9.2 logging

The Logging facilies outputs information, error and debugging information to the console or to a file. The level of verbosity can be configured at run time. The implementation includes facility methods to build complex debugging output strings, tracking time between calls, and output preformatted data such as vectors and matrices.

### A.9.3 input/output formats for images and matrices

The current implementation includes a generic interface for reading and writing images, and a specific implementation for the PGM format usually found as the "raw" image format under Unix or its variants. It also includes Windows Bitmap (BMP) read and write capability.

Also included are utilities to read and write Matlab (4.2) matrices, which are also handled by Scilab (an open source Matlab clone), to communicate data between these development tools and the C++ program.

## A.10 command line interface

As the whole implementation is modularized, the command line interface is just a small program which accepts the full set of configuration parameters through a configuration file and/or command line parameters, a noisy image to be denoised or else a clean image to add simulated noise and then denoise it, producing the denoised output as well as optional analysis information for experimentation purposes. Some of the common usage cases are described below.

**Without arguments** , the command

```
dude
```

runs a demo by creating a uniform 128 by 128 gray image, adding noise to it and then denoising it with the default configuration.

**To obtain online help** , type

```
dude -h
```

The execution will terminate immediatly. All the parameters are of the form `-key=value`, although the `-help` option shows the parameters without the hypen prefix.

**To obtain help for the full set of parameters** use `-X` before `-h`:

```
dude -X -h
```

**To simmulate noise on a clean image** and then denoise it using the default configuration:

```
dude some_image.pgm
```

This will use the default noise type and parameters (Salt and Pepper noise with $\lambda = 30\%$, Napkin modeler). The following example

```
   dude -channel=gaussian -channel.sigma=5 ...
-outdir=gauss5_test some_img.pgm
```

will corrupt the image using a Gaussian channel with $\sigma = 5$, denoise it and place the output in the `gauss5_test` directory.

**To create a default configuration file** `dude -create=name_of_the_file`

.

The ".cfg" extension has been adopted by convention, but it is not a requirement. Both the `-help` option and the generated configuration files give detailed information on each parameter and are a good source of information to learn how to use the program.

**To use a specific configuration file** ,

```
dude -config=some_cfg_file ...
```

**To avoid the addition of noise** (to clean an already noisy image), use

```
dude -add_noise=false ...
```

### A.10.1 configuration files

Configuration files are simple ASCII files where the lines are of the form `key=value` (whitout the preceding hypen). If a "#" appears on a file, the rest of the line in which it appears is ignored. Any line that begins with a "#" is considered a comment. The best way to use the command line interface is to produce a default configuration file with the `-create` command.

## A.11 graphical user interface

A graphical user interface (GUI) is included for ease of use. The GUI is written in the Java language as it is very easy to write such applications in that language and also highly portable as a way to produce graphical user interfaces.

The GUI is easier to use than the command line interface, although it doesn't give access to the full range of parameters. The interface shows a twin display which pans and zooms synchronously so that comparison between images is easy at any resolution or even pixel by pixel. The basic operations are presented as buttons in the main window, while the rest is contained in the menu bar.

The GUI also contains some basic tools for the analysis of the denoising process (image differences,standard measures such as PSNR, etc.). Finally, the rest of the configurable parameters that are not accesible can be set by creating a custom configuration file and loading it with the GUI (these files are the same used by the command line interface).

### A.11.1 Java/C++ integration (JNI)

The C++/Java communication is carried out using the JNI standard (Java Native Interface) mechanism that comes with the development kit. The GUI implementation is thus divided in a series of Java classes and a series of C files which interface the Java classes with the DUDE implementation. The compilation of such a program is rather complicated and requires the use of some specific Java tools to complete the process. These steps are included in both the GNU Make makefile and Visual C++ Solution file for the GUI so no real knowledge is needed to compile it, but certain special requirements are still needed. For instance, the *Java Development Kit*[3] (1.4 or above) is needed to compile the GUI, the `JAVA_HOME` environment variable must be defined, and the Java compilation tools (javac, javah) need to be included in the `PATH` environment variable.

[3] `http://javasoft.sun.com/`

# B Fast closed forms for the denoising function

In the following derivations it is assumed that the expected loss is computed with respect to a distribution $P$ over an alphabet $\mathcal{A} = \{0, \ldots, M-1\}$.

## B.1 For the $L_1$ loss model

Consider the expected loss for the $L_1$ error and a chosen denoiser output $\alpha$. In this case each term of the loss matrix $\Lambda_{x\alpha} = |x - \alpha|$ and the expected loss $R_\alpha$ can be written as

$$R_\alpha = \sum_{x=0}^{x=M-1} P(X = x)|x - \alpha| \tag{B.1}$$

this can be rewritten as

$$R_\alpha = \sum_{x=0}^{x=\alpha-1} P(X = x)(\alpha - x) + \sum_{x=\alpha+1}^{M-1} P(X = x)(x - \alpha) \tag{B.2}$$

Consider the definition of the median of $P$, $\alpha_{\text{med}}$ for which

1. $P(X \leq \alpha_{\text{med}}) \geq 1/2$

2. $P(X \geq \alpha_{\text{med}}) \geq 1/2$

To prove that $\alpha_{\text{med}}$ yields the minimum expected loss, it suffices to show that $R_\alpha$ is a monotonically decreasing function for $\alpha \leq \alpha_{\text{med}}$, and monotonically increasing for $\alpha \geq \alpha_{\text{med}}$. For this, take the difference $R_\alpha - R_{\alpha-1}$:

$$\begin{aligned}
R_\alpha - R_{\alpha-1} &= \sum_{x=0}^{x=\alpha-1} P(X=x)(\alpha-x) + \sum_{x=\alpha+1}^{M-1} P(X=x)(x-\alpha) - \\
&\quad \left(\sum_{x=0}^{x=\alpha-2} P(X=x)(\alpha-1-x) + \sum_{x=\alpha}^{M-1} P(X=x)(x-(\alpha-1))\right) \\
&= \sum_{x=0}^{x=\alpha-1} P(X=x)(\alpha-x) - \sum_{x=0}^{x=\alpha-2} P(X=x)(\alpha-1-x) + \\
&\quad \sum_{x=\alpha+1}^{M-1} P(X=x)(x-\alpha) - \sum_{x=\alpha}^{M-1} P(X=x)(x-(\alpha-1)) \\
&= \sum_{x=0}^{x=\alpha-1} P(X=x)(\alpha-x) - \sum_{x=0}^{x=\alpha-2} P(X=x)(\alpha-x) + \sum_{x=0}^{\alpha-2} P(X=x) + \\
&\quad \sum_{x=\alpha+1}^{M-1} P(X=x)(x-\alpha) - \sum_{x=\alpha}^{M-1} P(X=x)(x-\alpha) - \sum_{x=\alpha}^{M-1} P(X=x) \\
&= P(X=\alpha-1) + \sum_{x=0}^{\alpha-2} P(X=x) - \sum_{x=\alpha}^{M-1} P(X=x) \\
&= \sum_{x=0}^{\alpha-1} P(X=x) - \sum_{x=\alpha}^{M-1} P(X=x) \\
&= P(X \le \alpha-1) - P(X \ge \alpha) = 1 - 2P(X \ge \alpha)
\end{aligned} \tag{B.3}$$

Using the definition of $\alpha_{\text{med}}$,

$$R_\alpha - R_{\alpha-1} = \begin{cases} \le 0 & , \quad \alpha \ge \alpha_{\text{med}} \\ \ge 0 & , \quad \alpha \le \alpha_{\text{med}} \end{cases} \tag{B.4}$$

Thus, the global minimum is $\alpha = \alpha_{\text{med}}$.

## B.2 For the $L_2$ loss model

In this case $\Lambda_{x\alpha} = (x-\alpha)^2$. Using $E_P(.)$ to denote expectation over $P$,

$$R_\alpha = E_P[(x-\alpha)^2] \tag{B.5}$$

which can be developed using the basic properties of expectation

$$\begin{aligned}
R_\alpha &= E_P[(x^2 - 2\alpha x + \alpha^2] && \text{(B.6)} \\
&= E_P[x^2] - 2\alpha E_P + \alpha^2 && \\
&&& \text{(B.7)}
\end{aligned}$$

if $\alpha$ is relaxed to be a continuos value between 0 and $M-1$, $(x-\alpha)^2$ is a strictly convex function of $\alpha$ and a global optimum can be found by differentiating (B.7)

$$\frac{dR_\alpha}{d\alpha} = 2\alpha - 2E_P[x] \tag{B.8}$$

where the optimum corresponds to $\frac{dR_\alpha}{d\alpha}$, i.e., $\alpha = E_P$. When this optimum is not integer, some strategy is used to map it to an integer value within $\mathcal{A}$, for example, rounding.

# C Parametric second pass for Gaussian distributions

The purpose of this section is to obtain an expression for $\theta$, the parameter of the Two Sided Geometric Distribution, in terms of the variance of this distribution, $\sigma^2$. For this, consider a TSGD with parameter $\theta$ and mean 0,

$$P(X = x) = (1 - \theta)\theta^{|x|} \tag{C.1}$$

Its variance is given by the following expression:

$$\sigma^2 = (1 - \theta) \sum_{x=-\infty}^{x=\infty} \theta^{|x|} x^2 = 2(1 - \theta) \sum_{x=0}^{x=\infty} \theta^{|x|} x^2 \tag{C.2}$$

the series expanson of (C.2) yields

$$\begin{aligned}
\sigma^2 &= 2(1 - \theta)\frac{\theta(\theta + 1)}{\theta - 1^3} \\
\sigma^2 &= 2\frac{\theta(\theta + 1)}{\theta - 1^2} \\
\theta - 1^2\sigma^2 &= \theta(\theta + 1)
\end{aligned}$$

which is reordered to obtain a second order polynomial on $\theta$

$$(\sigma^2 - 2)\theta^2 - 2(\sigma^2 + 1) + \sigma^2 = 0 \tag{C.3}$$

and finally

$$\theta = \frac{\sigma^2 + 1}{\sigma^2 - 2} + -\frac{\sqrt{1 + 4\sigma^2}}{\sigma^2 - 2} \tag{C.4}$$

# D Full results

This appendix presents the full set of results obtained in the experiments. For the Legacy Modeling scheme, the set of images includes a subset of the images of the SIPI database (`http://sipi.usc.edu/services/database/`), but exludes the bigger images contained in the JPEG-LS test suite since the computational resources required were too much for the machines used to run the tests. The Napkin results were obtained for both test suites as the computational resources required for this scheme are much smaller.

Most of the images of the SIPI database are of about 1/4 million pixels ($512 \times 512$ or $720 \times 576$), excepting "Camera","us" and "house" whose size is $256 \times 256$ (four times smaller). This has an impact in the parameters which depend on the size of the image such as the number of context classes.

## D.1 Best results

| $\lambda$ | baboon | barb2 | barb | boats | bridge | camera | goldy | hotely |
|---|---|---|---|---|---|---|---|---|
| 10% | 32.8 | 37.1 | 39.2 | 42.1 | 35.0 | 33.8 | 41.3 | 40.3 |
| 30% | 27.7 | 32.9 | 33.9 | 36.7 | 30.0 | 30.0 | 36.0 | 35.2 |
| $\lambda$ | house | lena | peppers | splash | tulips | us | average | |
| 10% | 37.4 | 42.2 | 38.1 | 39.4 | 42.1 | 33.4 | 38.1 | |
| 30% | 34.7 | 37.5 | 33.8 | 39.3 | 37.2 | 28.6 | 33.8 | |

Table D.1: Best results for the Salt and Pepper channel. Legacy scheme.

| $\lambda$ | baboon | barb2 | barb | boats | bridge | camera | goldy | hotely |
|---|---|---|---|---|---|---|---|---|
| 10% | 33.5 | 39.4 | 38.7 | 45.3 | 36.1 | 36.7 | 43.1 | 42.9 |
| 30% | 27.8 | 32.9 | 31.7 | 38.3 | 30.6 | 31.1 | 36.9 | 36.3 |
| $\lambda$ | house | lena | peppers | splash | tulips | us | aerial2 | bike3 |
| 10% | 46.2 | 44.3 | 37.5 | 48.4 | 45.2 | 34.2 | 39.2 | 35.4 |
| 30% | 38.4 | 38.2 | 33.5 | 41.5 | 37.9 | 28.7 | 32.8 | 30.7 |
| $\lambda$ | bike | cafe | cats | tools | average | | | |
| 10% | 33.4 | 32.9 | 38.3 | 29.2 | 39.0 | | | |
| 30% | 29.6 | 27.6 | 31.8 | 24.5 | 33.0 | | | |

Table D.2: Best results for the Salt and Pepper channel. Napkin scheme.

| $\lambda$ | baboon | barb2 | barb | boats | bridge | camera | goldy | hotely |
|---|---|---|---|---|---|---|---|---|
| 70% | 21.9 | 25.3 | 24.9 | 30.3 | 24.4 | 24.4 | 29.2 | 28.3 |
| $\lambda$ | house | lena | peppers | splash | tulips | us | average | |
| 70% | 30.4 | 30.7 | 25.7 | 33.3 | 29.8 | 18.9 | 27.0 | |

Table D.3: Best results for extreme Salt and Pepper channel. Napkin scheme.

| $\lambda$ | baboon | barb2 | barb | boats | bridge | camera | goldy | hotely |
|---|---|---|---|---|---|---|---|---|
| 10% | 33.5 | 39.6 | 41.0 | 44.4 | 35.9 | 34.4 | 42.7 | 42.4 |
| 30% | 28.1 | 33.7 | 34.9 | 38.2 | 30.7 | 30.3 | 36.9 | 36.5 |
| $\lambda$ | house | lena | peppers | splash | tulips | us | average | |
| 10% | 38.0 | 43.0 | 40.1 | 44.5 | 44.5 | 33.6 | 39.8 | |
| 30% | 35.4 | 37.8 | 35.8 | 39.7 | 37.7 | 27.4 | 34.5 | |

Table D.4: Best results for the Salt and Pepper channel. Combined scheme.

| ¶$_e$ | baboon | barb2 | barb | boats | bridge | camera | goldy | hotely |
|---|---|---|---|---|---|---|---|---|
| 10% | 27.8 | 32.6 | 31.4 | 36.4 | 30.6 | 28.1 | 36.6 | 35.1 |
| 20% | 25.3 | 29.3 | 28.4 | 33.0 | 28.1 | 26.0 | 33.5 | 31.3 |
| 30% | 23.6 | 26.9 | 26.4 | 30.5 | 26.3 | 24.5 | 30.9 | 28.3 |
| $P_e$ | house | lena | peppers | splash | tulips | us | aerial2 | bike3 |
| 10% | 33.9 | 37.1 | 36.1 | 38.1 | 36.2 | 24.5 | 31.2 | 29.4 |
| 20% | 31.8 | 34.2 | 32.9 | 35.9 | 32.8 | 22.1 | 28.3 | 27.2 |
| 30% | 30.4 | 31.7 | 30.2 | 33.5 | 30.0 | 19.7 | 25.6 | 25.4 |
| $P_e$ | bike | cafe | cats | tools | average | | | |
| 10% | 27.2 | 26.1 | 32.8 | 24.7 | 31.3 | | | |
| 20% | 25.2 | 23.6 | 29.5 | 21.9 | 28.6 | | | |
| 30% | 23.9 | 21.9 | 27.0 | 20.4 | 26.4 | | | |

Table D.5: Best results for the $q$-ary symmetric channel. Napkin scheme.

| $\sigma$ | baboon | barb2 | barb | boats | bridge | camera | goldy | hotely |
|---|---|---|---|---|---|---|---|---|
| 10 | 29.9 | 30.8 | 32.4 | 34.0 | 30.3 | 32.5 | 32.9 | 33.4 |
| 20 | 25.6 | 26.6 | 28.6 | 30.4 | 26.4 | 28.5 | 29.7 | 29.8 |
| 25 | 24.4 | 25.7 | 27.3 | 29.3 | 25.3 | 27.4 | 28.7 | 28.6 |
| $\sigma$ | house | lena | peppers | splash | tulips | us | average | |
| 10 | 34.1 | 34.2 | 34.6 | 34.7 | 33.6 | 32.4 | 32.8 | |
| 20 | 31.1 | 31.0 | 31.3 | 30.4 | 30.4 | 29.3 | 29.2 | |
| 25 | 29.8 | 29.9 | 30.6 | 29.4 | 29.3 | 27.8 | 28.1 | |

Table D.6: Best results for the Gaussian channel. Legacy modeling scheme.

## D.2 Selection of the parameters

### D.2.1 Legacy for Salt and Pepper

| NC | baboon | barb2 | barb | boats | bridge | camera | goldy | hotely |
|---|---|---|---|---|---|---|---|---|
| 64 | 5.9 | 4.8 | 4.8 | 4.0 | 5.5 | 4.3 | 4.5 | 4.4 |
| 128 | 5.9 | 4.7 | 4.8 | 4.0 | 5.5 | 4.3 | 4.5 | 4.4 |
| 192 | 5.9 | 4.7 | 4.8 | 4.0 | 5.5 | 4.4 | 4.4 | 4.4 |
| 256 | 5.9 | 4.7 | 4.8 | 4.0 | 5.5 | 4.5 | 4.5 | 4.4 |
| 320 | 5.9 | 4.7 | 4.8 | 4.0 | 5.5 | 4.5 | 4.5 | 4.4 |
| NC | house | lena | peppers | splash | tulips | us | average | |
| 64 | 4.1 | 4.2 | 3.9 | 3.8 | 4.3 | 3.0 | 4.4 | |
| 128 | 4.2 | 4.2 | 3.9 | 3.8 | 4.3 | 3.0 | 4.4 | |
| 192 | 4.2 | 4.2 | 4.0 | 3.9 | 4.3 | 3.1 | 4.4 | |
| 256 | 4.4 | 4.2 | 4.0 | 3.9 | 4.3 | 3.1 | 4.4 | |
| 320 | 4.4 | 4.2 | 4.0 | 4.0 | 4.3 | 3.1 | 4.5 | |

Table D.7: Legacy for Salt and Pepper. Compressibility vs. number of context clusters.

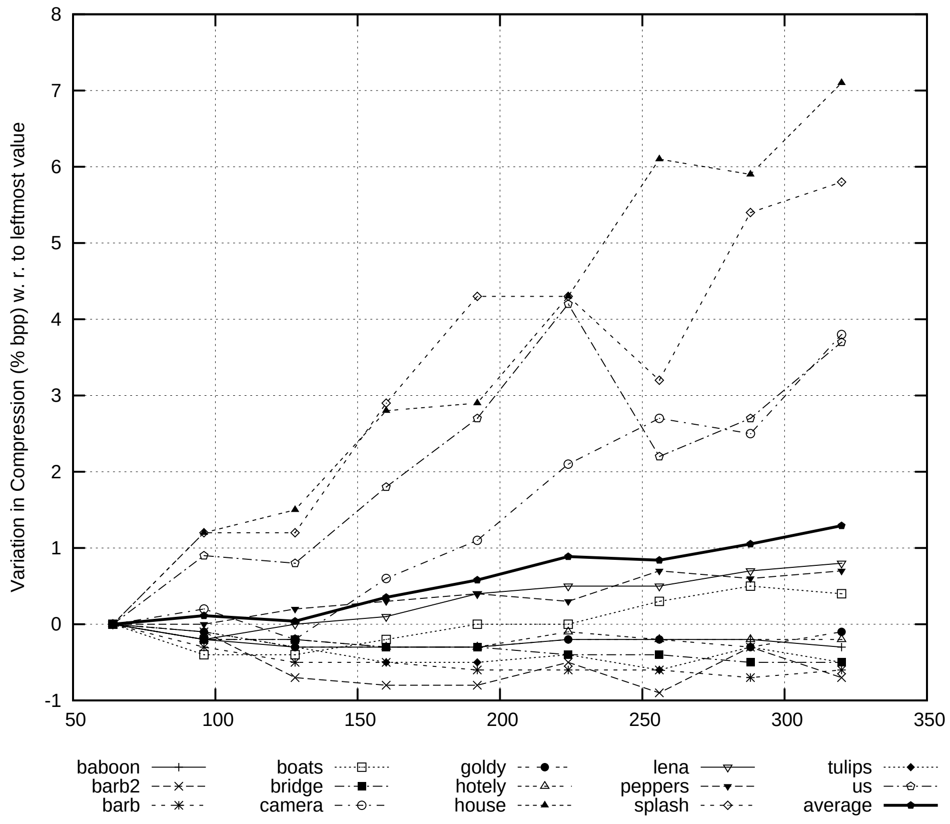


Figure D.1: Legacy for Salt and Pepper. Compressibility vs. number of context clusters.

| NC | baboon | barb2 | barb | boats | bridge | camera | goldy | hotely |
|---|---|---|---|---|---|---|---|---|
| 64 | 26.9 | 30.9 | 31.1 | 35.1 | 29.4 | 29.6 | 35.1 | 33.5 |
| 128 | 27.2 | 31.1 | 31.8 | 35.4 | 29.5 | 29.9 | 35.3 | 33.9 |
| 192 | 27.2 | 31.2 | 31.9 | 35.4 | 29.6 | 29.5 | 35.3 | 33.9 |
| 256 | 27.2 | 31.3 | 32.1 | 35.5 | 29.6 | 29.3 | 35.3 | 34.0 |
| 320 | 27.2 | 31.3 | 32.2 | 35.5 | 29.6 | 29.2 | 35.3 | 34.0 |
| NC | house | lena | peppers | splash | tulips | us | average | |
| 64 | 34.7 | 36.5 | 32.2 | 38.7 | 35.8 | 26.9 | 32.6 | |
| 128 | 34.8 | 36.6 | 32.6 | 38.4 | 36.1 | 26.8 | 32.8 | |
| 192 | 34.1 | 36.7 | 32.6 | 37.8 | 36.2 | 27.1 | 32.7 | |
| 256 | 33.3 | 36.7 | 32.4 | 38.0 | 36.3 | 27.1 | 32.7 | |
| 320 | 33.2 | 36.6 | 32.5 | 37.1 | 36.3 | 27.0 | 32.6 | |

Table D.8: Legacy for Salt and Pepper. PSNR vs. number of context clusters.

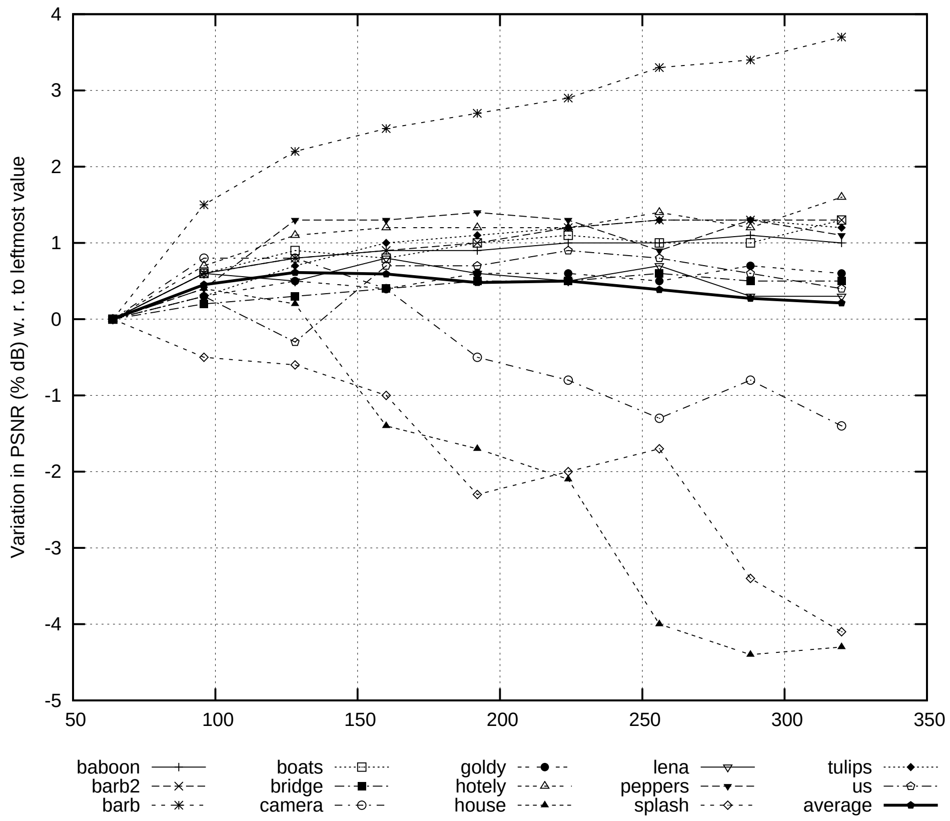


Figure D.2: Legacy for Salt and Pepper. PSNR vs. number of context clusters.

| Context | baboon | barb2 | barb | boats | bridge | camera | goldy | hotely |
|---|---|---|---|---|---|---|---|---|
| $3 \times 3$ | 6.0 | 4.9 | 4.9 | 4.1 | 5.6 | 4.6 | 4.5 | 4.5 |
| $5 \times 5$ | 5.9 | 4.7 | 4.8 | 4.0 | 5.5 | 4.5 | 4.5 | 4.4 |
| $7 \times 7$ | 6.0 | 4.8 | 4.8 | 4.1 | 5.6 | 4.5 | 4.5 | 4.5 |
| Context | house | lena | peppers | splash | tulips | us | average | |
| $3 \times 3$ | 4.4 | 4.2 | 4.1 | 3.9 | 4.4 | 3.1 | 4.5 | |
| $5 \times 5$ | 4.4 | 4.2 | 4.0 | 3.9 | 4.3 | 3.1 | 4.4 | |
| $7 \times 7$ | 4.4 | 4.2 | 4.0 | 3.8 | 4.3 | 3.5 | 4.5 | |

Table D.9: Legacy for Salt and Pepper. Compressibility vs. radius of the contextss.

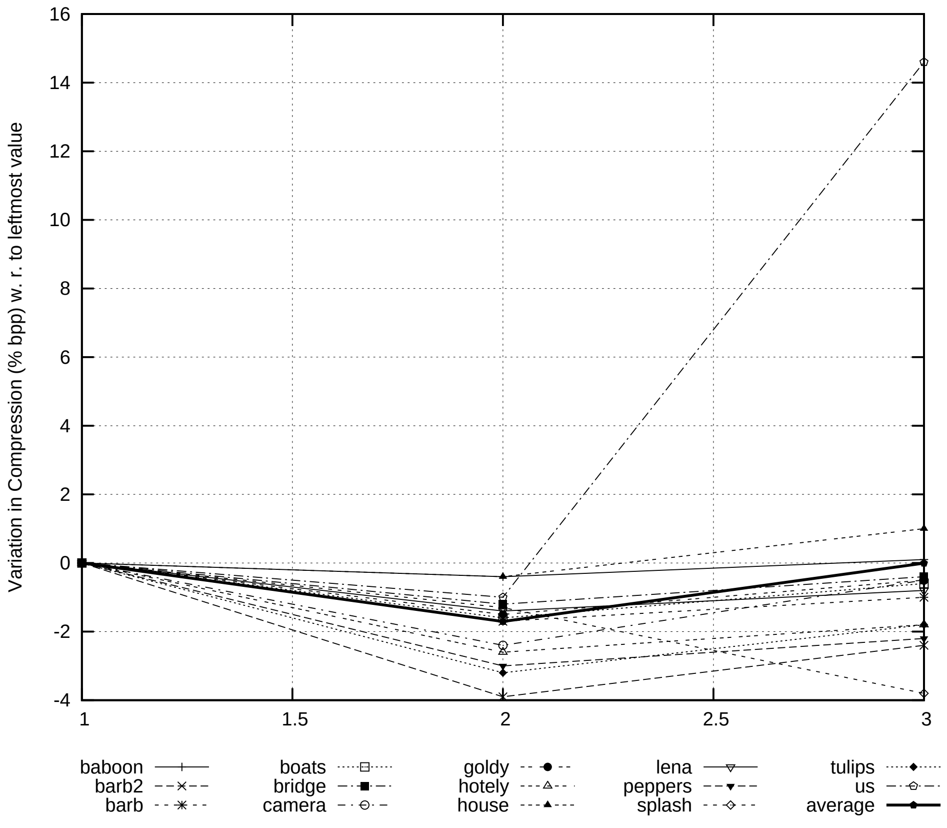


Figure D.3: Legacy for Salt and Pepper. Compressibility vs. radius of the contexts.

| Context | baboon | barb2 | barb | boats | bridge | camera | goldy | hotely |
|---|---|---|---|---|---|---|---|---|
| 3 × 3 | 26.7 | 29.7 | 29.9 | 33.7 | 28.8 | 28.1 | 33.9 | 31.4 |
| 5 × 5 | 27.2 | 31.3 | 32.1 | 35.5 | 29.6 | 29.3 | 35.3 | 34.0 |
| 7 × 7 | 26.8 | 31.2 | 32.1 | 35.0 | 29.1 | 28.9 | 34.7 | 33.2 |
| Context | house | lena | peppers | splash | tulips | us | average | |
| 3 × 3 | 32.1 | 34.9 | 31.5 | 37.0 | 34.1 | 26.2 | 31.3 | |
| 5 × 5 | 33.3 | 36.7 | 32.4 | 38.0 | 36.3 | 27.1 | 32.7 | |
| 7 × 7 | 32.9 | 36.1 | 31.7 | 38.6 | 35.5 | 26.7 | 32.3 | |

Table D.10: Legacy for Salt and Pepper. PSNR vs. radius of the contexts.

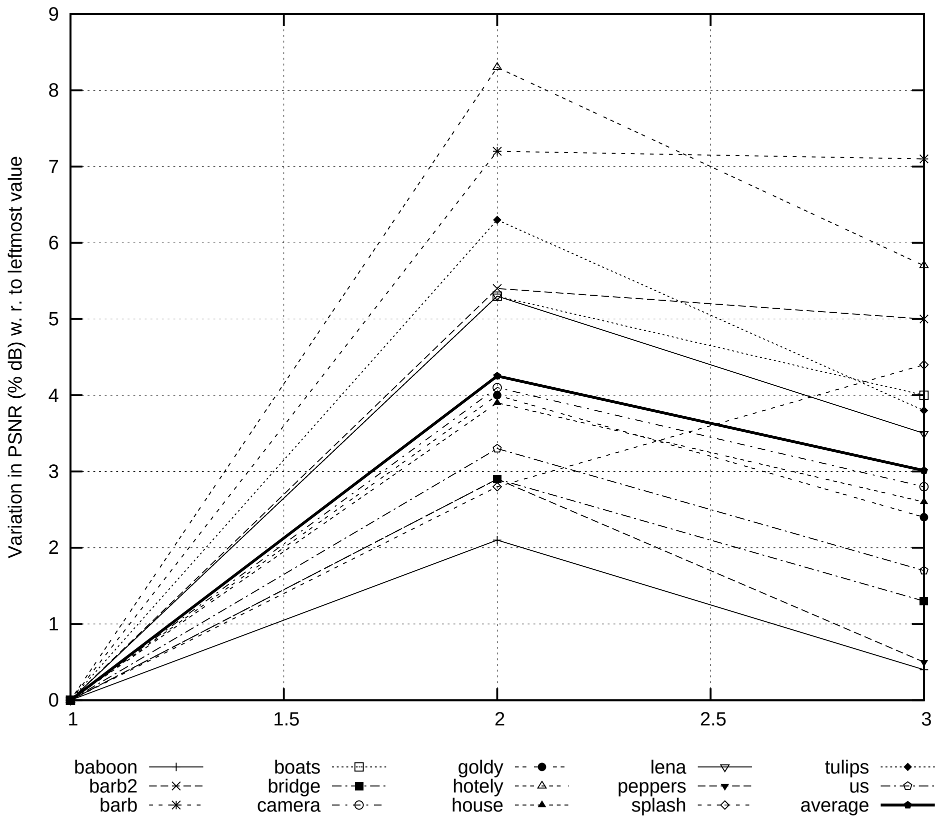


Figure D.4: Legacy for Salt and Pepper. PSNR vs. radius of the contexts.

| Iter. | baboon | barb2 | barb | boats | bridge | camera | goldy | hotely |
|---|---|---|---|---|---|---|---|---|
| 1.00 | 5.9 | 4.7 | 4.7 | 4.0 | 5.5 | 4.5 | 4.4 | 4.4 |
| 2.00 | 5.9 | 4.7 | 4.7 | 4.0 | 5.5 | 4.5 | 4.4 | 4.3 |
| 3.00 | 5.9 | 4.7 | 4.7 | 4.0 | 5.5 | 4.4 | 4.4 | 4.4 |
| 4.00 | 5.9 | 4.7 | 4.7 | 4.0 | 5.5 | 4.4 | 4.4 | 4.4 |
| 5.00 | 5.9 | 4.7 | 4.7 | 4.0 | 5.5 | 4.5 | 4.4 | 4.3 |
| 6.00 | 5.9 | 4.7 | 4.7 | 4.0 | 5.5 | 4.5 | 4.4 | 4.4 |
| Iter. | house | lena | peppers | splash | tulips | us | average | |
| 1 | 4.4 | 4.2 | 3.9 | 3.8 | 4.2 | 3.1 | 4.4 | |
| 2 | 4.3 | 4.2 | 4.0 | 3.7 | 4.2 | 3.1 | 4.4 | |
| 3 | 4.3 | 4.2 | 4.0 | 3.8 | 4.2 | 3.0 | 4.4 | |
| 4 | 4.4 | 4.2 | 3.9 | 3.8 | 4.2 | 3.1 | 4.4 | |
| 5 | 4.3 | 4.2 | 3.9 | 3.8 | 4.2 | 3.1 | 4.4 | |
| 6 | 4.3 | 4.2 | 3.9 | 3.8 | 4.2 | 3.0 | 4.4 | |

Table D.11: Legacy for Salt and Pepper. Compressibility vs. number of prefiltering iterations

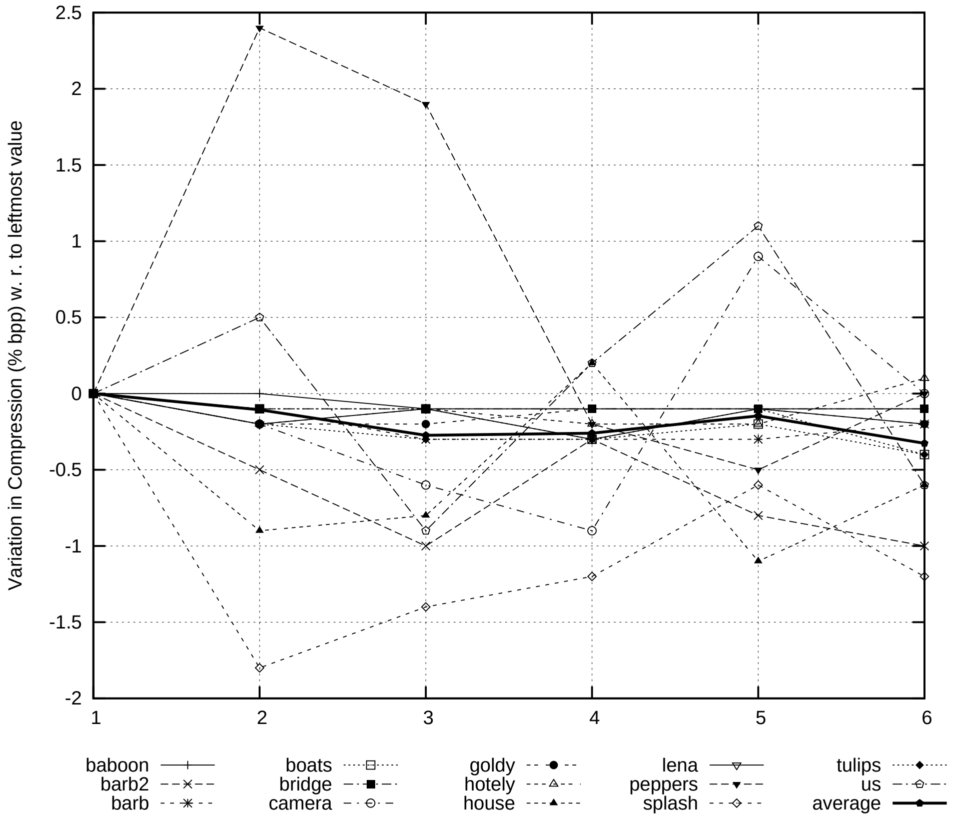


Figure D.5: Legacy for Salt and Pepper. Compressibility vs. number of prefiltering iterations

| Iter. | baboon | barb2 | barb | boats | bridge | camera | goldy | hotely |
|---|---|---|---|---|---|---|---|---|
| 1 | 27.6 | 32.4 | 33.4 | 36.4 | 30.0 | 29.7 | 35.8 | 34.9 |
| 2 | 27.7 | 32.7 | 33.7 | 36.6 | 30.0 | 30.1 | 36.0 | 35.2 |
| 3 | 27.7 | 32.9 | 33.9 | 36.7 | 30.1 | 30.0 | 36.0 | 35.2 |
| 4 | 27.7 | 32.9 | 33.9 | 36.7 | 30.0 | 30.0 | 36.0 | 35.2 |
| 5 | 27.7 | 32.9 | 33.9 | 36.7 | 30.0 | 30.1 | 36.0 | 35.2 |
| 6 | 27.7 | 32.9 | 33.8 | 36.7 | 30.0 | 30.2 | 36.0 | 35.1 |
| **Iter.** | **house** | **lena** | **peppers** | **splash** | **tulips** | **us** | **average** | |
| 1 | 34.2 | 37.3 | 33.4 | 38.6 | 37.0 | 28.3 | 33.5 | |
| 2 | 34.8 | 37.5 | 33.3 | 39.4 | 37.2 | 28.5 | 33.8 | |
| 3 | 35.2 | 37.4 | 33.6 | 39.4 | 37.2 | 28.4 | 33.8 | |
| 4 | 34.7 | 37.5 | 33.8 | 39.3 | 37.2 | 28.6 | 33.8 | |
| 5 | 34.6 | 37.4 | 33.7 | 39.1 | 37.2 | 28.6 | 33.8 | |
| 6 | 34.9 | 37.5 | 33.5 | 39.4 | 37.2 | 28.5 | 33.8 | |

Table D.12: Legacy for Salt and Pepper. PSNR vs. number of prefiltering iterations.

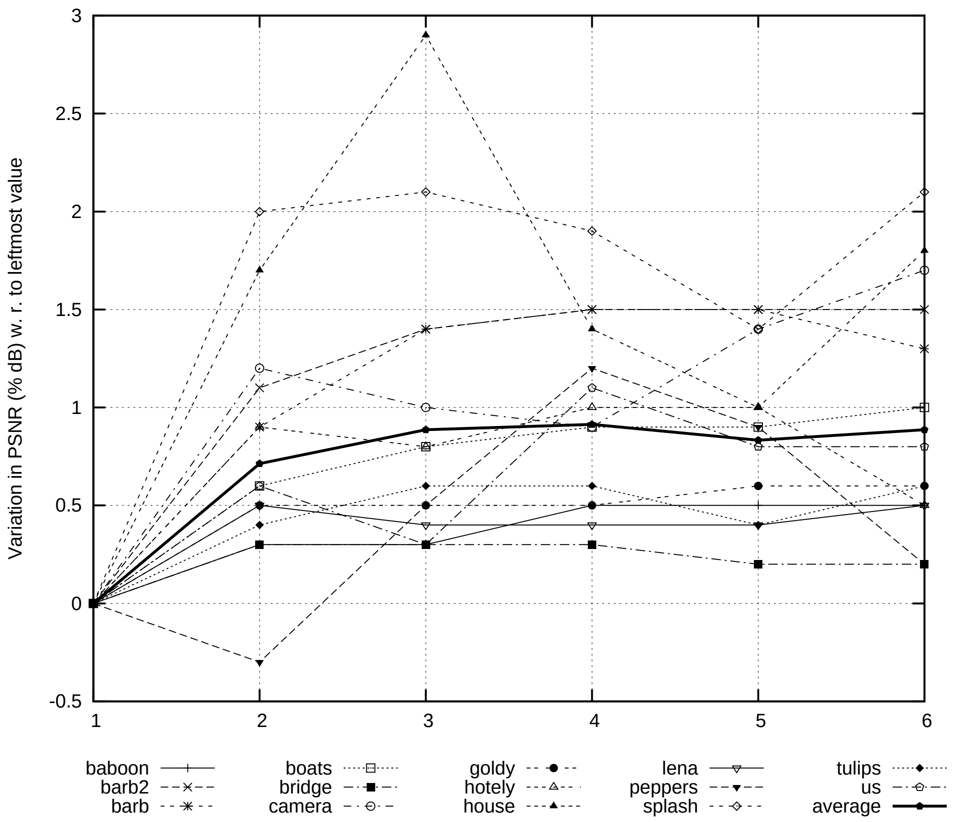


Figure D.6: Legacy for Salt and Pepper. PSNR vs. number of prefiltering iterations

| Iter. | baboon | barb2 | barb | boats | bridge | camera | goldy | hotely |
|---|---|---|---|---|---|---|---|---|
| 2 | 5.9 | 4.8 | 4.8 | 4.0 | 5.5 | 4.6 | 4.5 | 4.4 |
| 4 | 5.9 | 4.8 | 4.8 | 4.0 | 5.5 | 4.5 | 4.5 | 4.4 |
| 8 | 5.9 | 4.8 | 4.8 | 4.0 | 5.5 | 4.5 | 4.5 | 4.4 |
| 20 | 5.9 | 4.7 | 4.8 | 4.0 | 5.5 | 4.5 | 4.5 | 4.4 |
| 40 | 5.9 | 4.7 | 4.8 | 4.0 | 5.5 | 4.4 | 4.5 | 4.4 |
| 80 | 5.9 | 4.7 | 4.8 | 4.0 | 5.5 | 4.4 | 4.5 | 4.4 |
| Iter. | house | lena | peppers | splash | tulips | us | average | |
| 2 | 4.4 | 4.2 | 4.0 | 3.9 | 4.3 | 3.2 | 4.5 | |
| 4 | 4.4 | 4.2 | 4.0 | 3.9 | 4.3 | 3.2 | 4.5 | |
| 8 | 4.4 | 4.2 | 4.0 | 3.9 | 4.3 | 3.2 | 4.4 | |
| 20 | 4.4 | 4.2 | 4.0 | 3.9 | 4.3 | 3.1 | 4.4 | |
| 40 | 4.4 | 4.2 | 4.0 | 3.9 | 4.3 | 3.1 | 4.4 | |
| 80 | 4.3 | 4.2 | 4.0 | 3.9 | 4.3 | 3.0 | 4.4 | |

Table D.13: Legacy for Salt and Pepper. Comp. vs. number of LBG iterations.

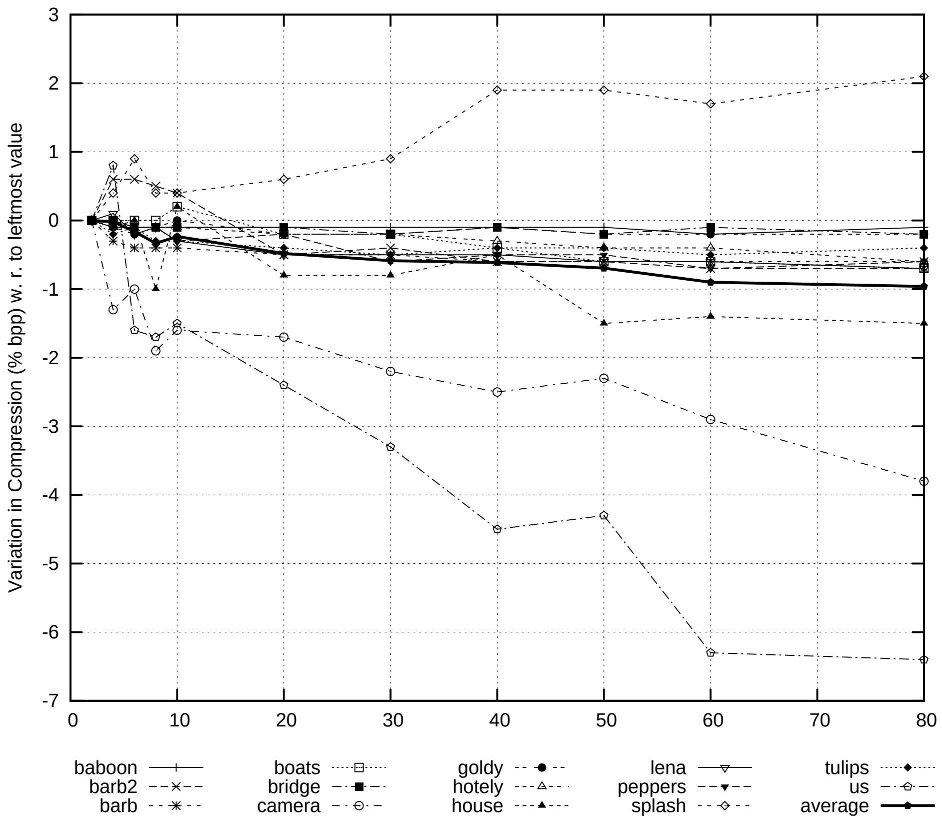


Figure D.7: Legacy for Salt and Pepper. Comp. vs. number of LBG iterations.

| Iter. | baboon | barb2 | barb | boats | bridge | camera | goldy | hotely |
|---|---|---|---|---|---|---|---|---|
| 2 | 27.0 | 30.9 | 31.4 | 35.2 | 29.5 | 29.2 | 35.1 | 33.8 |
| 4 | 27.1 | 31.0 | 31.8 | 35.2 | 29.6 | 29.3 | 35.2 | 33.8 |
| 8 | 27.2 | 31.2 | 31.9 | 35.3 | 29.6 | 29.3 | 35.3 | 33.9 |
| 20 | 27.2 | 31.2 | 32.1 | 35.5 | 29.6 | 29.3 | 35.3 | 34.0 |
| 40 | 27.2 | 31.3 | 32.1 | 35.5 | 29.6 | 29.2 | 35.3 | 34.0 |
| 80 | 27.2 | 31.3 | 32.2 | 35.6 | 29.6 | 29.3 | 35.3 | 34.0 |
| Iter. | house | lena | peppers | splash | tulips | us | average | |
| 2 | 33.8 | 36.4 | 32.0 | 38.4 | 35.9 | 26.6 | 32.5 | |
| 4 | 33.8 | 36.5 | 32.2 | 38.4 | 36.1 | 26.9 | 32.6 | |
| 8 | 33.8 | 36.6 | 32.2 | 38.3 | 36.2 | 27.0 | 32.7 | |
| 20 | 33.6 | 36.7 | 32.3 | 38.1 | 36.3 | 27.0 | 32.7 | |
| 40 | 33.5 | 36.7 | 32.7 | 37.5 | 36.3 | 27.1 | 32.7 | |
| 80 | 33.5 | 36.7 | 32.7 | 37.4 | 36.4 | 27.2 | 32.7 | |

Table D.14: Legacy for Salt and Pepper. PSNR vs. number of LBG iterations.

| Pred. | baboon | barb2 | barb | boats | bridge | camera | goldy | hotely |
|---|---|---|---|---|---|---|---|---|
| average | 6.0 | 4.9 | 4.9 | 4.2 | 5.6 | 4.6 | 4.6 | 4.6 |
| median | 6.0 | 4.8 | 4.9 | 4.2 | 5.6 | 4.6 | 4.6 | 4.5 |
| gaussian | 6.0 | 4.8 | 4.9 | 4.2 | 5.5 | 4.6 | 4.6 | 4.5 |
| Pred. | house | lena | peppers | splash | tulips | us | average | |
| average | 4.4 | 4.3 | 4.2 | 4.0 | 4.5 | 4.1 | 4.6 | |
| median | 4.4 | 4.3 | 4.2 | 3.9 | 4.5 | 4.1 | 4.6 | |
| gaussian | 4.4 | 4.3 | 4.1 | 4.0 | 4.4 | 4.0 | 4.6 | |

Table D.15: Legacy for Salt and Pepper. Comp. vs. type of predictor.

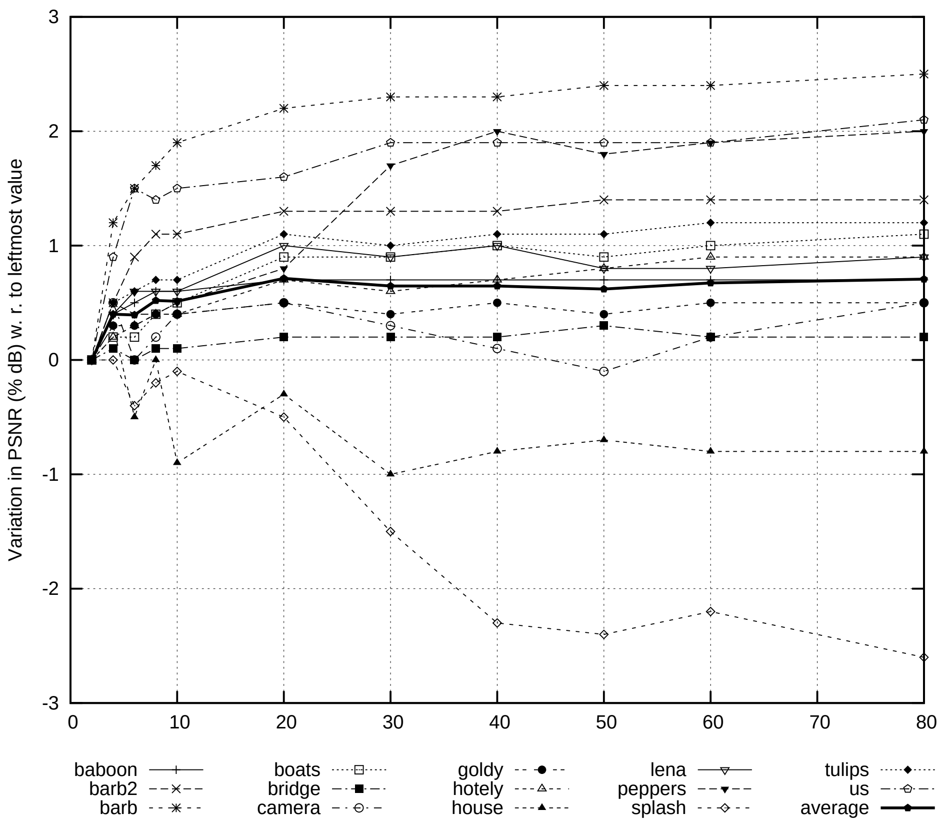


Figure D.8: Legacy for Salt and Pepper. PSNR vs. number of LBG iterations.

| Pred. | baboon | barb2 | barb | boats | bridge | camera | goldy | hotely |
|---|---|---|---|---|---|---|---|---|
| average | 27.2 | 31.3 | 32.1 | 35.5 | 29.6 | 29.3 | 35.3 | 34.0 |
| median | 26.9 | 30.8 | 31.7 | 35.0 | 29.4 | 29.0 | 35.1 | 33.4 |
| gaussian | 27.3 | 31.4 | 32.2 | 35.7 | 29.8 | 29.4 | 35.4 | 34.2 |
| Pred. | house | lena | peppers | splash | tulips | us | average | |
| average | 33.3 | 36.7 | 32.4 | 38.0 | 36.3 | 27.1 | 32.7 | |
| median | 32.9 | 36.4 | 32.0 | 38.2 | 35.9 | 26.4 | 32.4 | |
| gaussian | 33.5 | 36.9 | 32.5 | 38.5 | 36.6 | 27.2 | 32.9 | |

Table D.16: Legacy for Salt and Pepper. PSNR vs. type of predictor.

### D.2.2 Napkin for Salt and Pepper

| Iter. | baboon | barb2 | barb | boats | bridge | camera | goldy | hotely |
|---|---|---|---|---|---|---|---|---|
| 0 | 6.1 | 5.5 | 5.6 | 5.1 | 6.0 | 5.5 | 5.3 | 5.4 |
| 10 | 5.7 | 4.7 | 4.8 | 3.9 | 5.4 | 4.3 | 4.7 | 4.4 |
| 20 | 5.7 | 4.6 | 4.7 | 3.8 | 5.3 | 4.4 | 4.5 | 4.2 |
| 30 | 5.6 | 4.7 | 4.6 | 3.8 | 5.3 | 4.4 | 4.4 | 4.2 |
| 40 | 5.6 | 4.5 | 4.6 | 3.8 | 5.3 | 4.3 | 4.3 | 4.1 |
| 60 | 5.7 | 4.8 | 4.6 | 3.8 | 5.3 | 4.2 | 4.2 | 4.2 |
| 80 | 5.6 | 4.6 | 4.6 | 3.7 | 5.3 | 4.2 | 4.2 | 4.2 |
| Iter. | house | lena | peppers | splash | tulips | us | average | |
| 0 | 5.4 | 5.2 | 5.1 | 5.0 | 5.4 | 5.3 | 5.4 | |
| 10 | 4.4 | 4.2 | 3.9 | 3.8 | 4.4 | 4.4 | 4.5 | |
| 20 | 4.3 | 4.1 | 3.8 | 3.6 | 4.2 | 3.8 | 4.4 | |
| 30 | 4.4 | 4.1 | 3.8 | 3.4 | 4.2 | 3.6 | 4.3 | |
| 40 | 4.2 | 4.1 | 3.8 | 3.4 | 4.2 | 3.4 | 4.3 | |
| 60 | 4.2 | 4.0 | 3.7 | 3.4 | 4.2 | 3.3 | 4.3 | |
| 80 | 4.1 | 3.9 | 3.7 | 3.4 | 4.2 | 3.5 | 4.2 | |

Table D.17: Napkin for Extreme Salt and Pepper. Compressibility vs. prefiltering iterations.

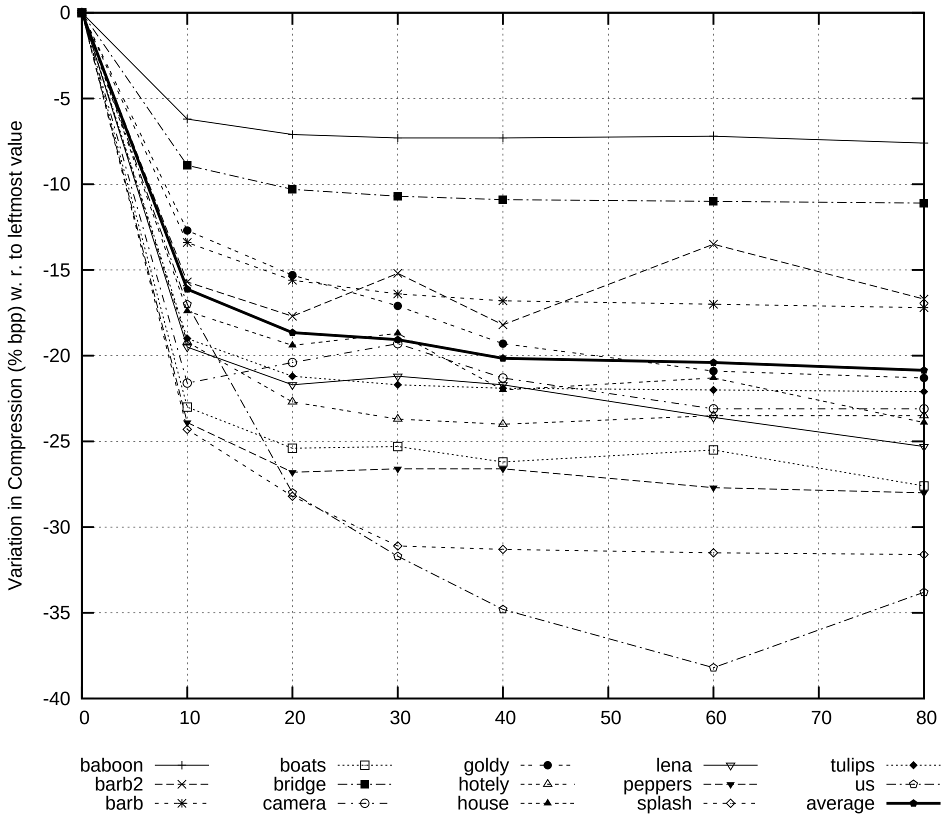


Figure D.9: Napkin for Extreme Salt and Pepper. Compressibility vs. prefiltering iterations.

| Iter. | baboon | barb2 | barb | boats | bridge | camera | goldy | hotely |
|---|---|---|---|---|---|---|---|---|
| 0 | 18.8 | 19.9 | 19.8 | 21.3 | 19.5 | 18.9 | 20.6 | 19.5 |
| 10 | 21.5 | 24.4 | 23.2 | 28.2 | 23.5 | 23.6 | 22.9 | 25.6 |
| 20 | 21.8 | 25.1 | 24.2 | 29.6 | 24.2 | 24.1 | 24.5 | 27.6 |
| 30 | 21.9 | 25.2 | 24.6 | 30.0 | 24.3 | 24.2 | 26.1 | 28.1 |
| 40 | 21.9 | 25.3 | 24.7 | 30.1 | 24.4 | 24.3 | 27.7 | 28.2 |
| 60 | 21.9 | 25.3 | 24.9 | 30.3 | 24.4 | 24.4 | 29.2 | 28.3 |
| 80 | 21.9 | 25.3 | 24.9 | 30.3 | 24.4 | 24.3 | 29.7 | 28.3 |
| Iter. | house | lena | peppers | splash | tulips | us | average | |
| 0 | 20.8 | 21.6 | 20.5 | 20.8 | 20.3 | 15.5 | 19.9 | |
| 10 | 28.2 | 29.0 | 25.7 | 26.3 | 27.7 | 17.5 | 24.8 | |
| 20 | 29.6 | 30.4 | 26.2 | 30.1 | 29.6 | 18.5 | 26.1 | |
| 30 | 29.9 | 30.6 | 26.1 | 32.9 | 29.8 | 18.9 | 26.6 | |
| 40 | 30.3 | 30.7 | 26.0 | 33.3 | 29.9 | 19.1 | 26.8 | |
| 60 | 30.4 | 30.7 | 25.7 | 33.3 | 29.8 | 18.9 | 27.0 | |
| 80 | 30.5 | 30.7 | 25.6 | 33.3 | 29.9 | 18.2 | 27.0 | |

Table D.18: Napkin for Extreme Salt and Pepper. PSNR vs. prefiltering iterations.

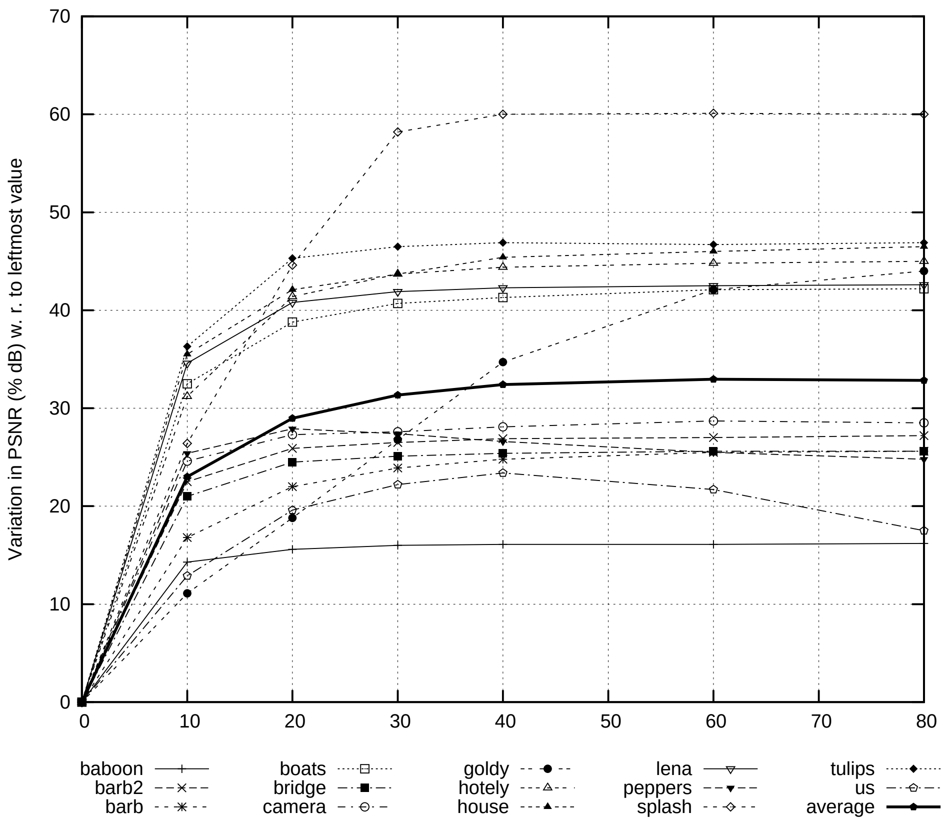


Figure D.10: Napkin for Extreme Salt and Pepper. PSNR vs. prefiltering iterations.

| ALB | baboon | barb2 | barb | boats | bridge | camera | goldy | hotely |
|---|---|---|---|---|---|---|---|---|
| 1 | 5.9 | 4.7 | 4.7 | 3.9 | 5.4 | 4.3 | 4.3 | 4.3 |
| 3 | 5.9 | 4.7 | 4.7 | 3.9 | 5.4 | 4.3 | 4.3 | 4.3 |
| 5 | 5.9 | 4.7 | 4.7 | 3.9 | 5.4 | 4.3 | 4.3 | 4.3 |
| 7 | 6.0 | 4.8 | 4.8 | 4.0 | 5.5 | 4.4 | 4.4 | 4.4 |
| ALB | house | lena | peppers | splash | tulips | us | aerial2 | bike3 |
| 1 | 4.0 | 4.1 | 3.8 | 3.5 | 4.2 | 3.0 | 4.1 | 4.2 |
| 3 | 4.0 | 4.1 | 3.8 | 3.5 | 4.1 | 2.9 | 4.1 | 4.2 |
| 5 | 4.0 | 4.1 | 3.8 | 3.5 | 4.1 | 3.0 | 4.1 | 4.2 |
| 7 | 4.2 | 4.2 | 3.9 | 3.6 | 4.2 | 3.2 | 4.2 | 4.2 |
| ALB | bike | cafe | cats | tools | average | | | |
| 1 | 4.2 | 5.0 | 5.1 | 5.4 | 4.4 | | | |
| 3 | 4.1 | 5.0 | 5.0 | 5.3 | 4.4 | | | |
| 5 | 4.1 | 5.0 | 5.0 | 5.3 | 4.4 | | | |
| 7 | 4.1 | 5.0 | 5.1 | 5.3 | 4.5 | | | |

Table D.19: Napkin for Salt and Pepper. Compressibility vs. prefiltering number of context classes. The number of context classes is $2^{ALB}$ where $ALB$ are the activity level bits.

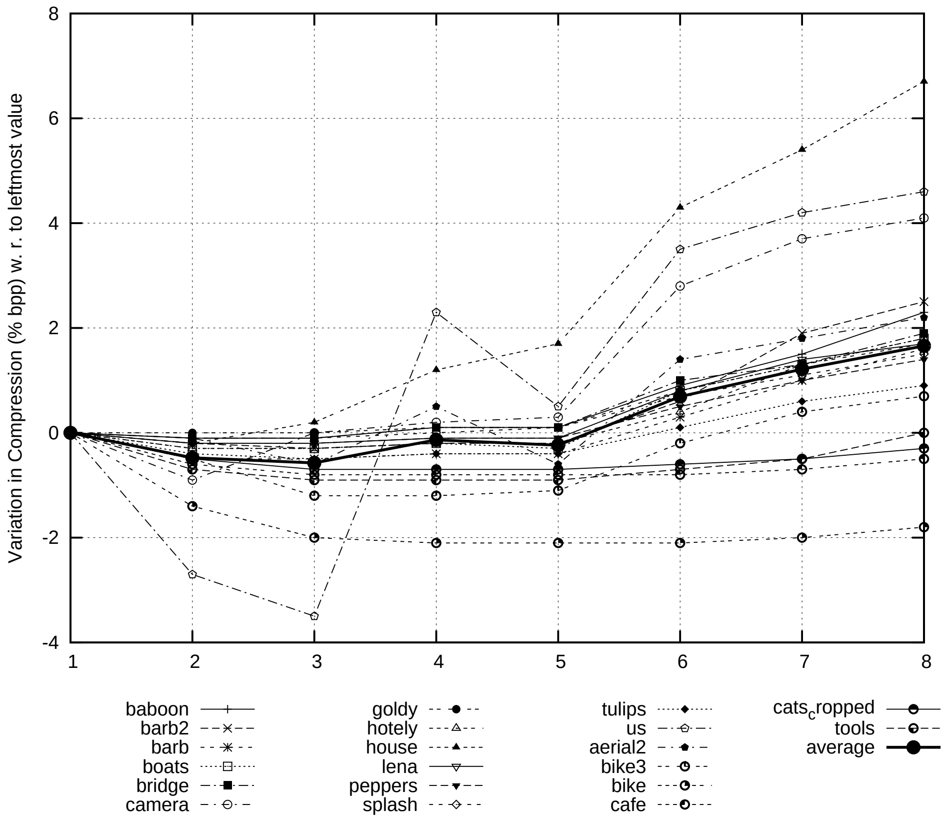


Figure D.11: Napkin for Salt and Pepper. Compressibility vs. prefiltering number of context classes.The number of context classes is $2^{ALB}$ where $ALB$ are the activity level bits.

| ALB | baboon | barb2 | barb | boats | bridge | camera | goldy | hotely |
|---|---|---|---|---|---|---|---|---|
| 1.00 | 27.4 | 30.8 | 30.7 | 35.9 | 30.0 | 30.1 | 35.8 | 33.8 |
| 3.00 | 27.4 | 30.9 | 30.8 | 36.0 | 30.0 | 30.0 | 35.8 | 33.9 |
| 5.00 | 27.4 | 30.9 | 30.8 | 35.9 | 30.0 | 29.9 | 35.8 | 33.9 |
| 7.00 | 26.8 | 30.4 | 30.2 | 35.5 | 29.5 | 29.2 | 35.4 | 33.6 |
| ALB | house | lena | peppers | splash | tulips | us | aerial2 | bike3 |
| 1 | 35.7 | 36.8 | 32.4 | 39.6 | 36.4 | 27.3 | 31.8 | 29.9 |
| 3 | 35.8 | 36.9 | 32.4 | 39.7 | 36.6 | 27.4 | 31.9 | 30.1 |
| 5 | 35.6 | 36.9 | 32.3 | 39.7 | 36.6 | 27.4 | 32.0 | 30.2 |
| 7 | 34.8 | 36.3 | 32.1 | 39.5 | 36.2 | 27.2 | 31.7 | 30.0 |
| ALB | bike | cafe | cats | tools | average | | | |
| 1 | 28.3 | 26.4 | 31.1 | 23.4 | 31.7 | | | |
| 3 | 28.4 | 26.5 | 31.3 | 23.6 | 31.8 | | | |
| 5 | 28.5 | 26.5 | 31.3 | 23.6 | 31.8 | | | |
| 7 | 28.5 | 26.5 | 31.2 | 23.5 | 31.4 | | | |

Table D.20: Napkin for Salt and Pepper. PSNR vs. prefiltering number of context classes.The number of context classes is $2^{ALB}$ where $ALB$ are the activity level bits.

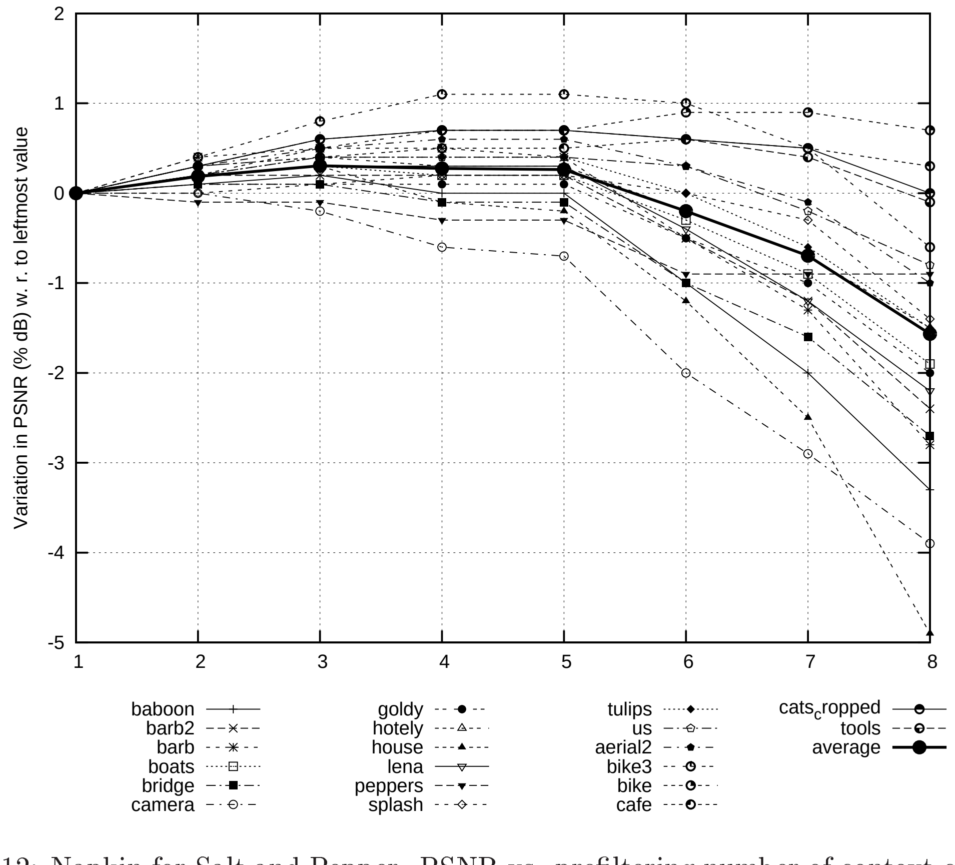


Figure D.12: Napkin for Salt and Pepper. PSNR vs. prefiltering number of context classes.The number of context classes is $2^{ALB}$ where $ALB$ are the activity level bits.

| Pred. | baboon | barb2 | barb | boats | bridge | camera | goldy | hotely |
|---|---|---|---|---|---|---|---|---|
| average | 5.6 | 4.4 | 4.4 | 3.7 | 5.1 | 4.0 | 4.1 | 4.0 |
| sharp | 5.6 | 4.4 | 4.5 | 3.8 | 5.2 | 4.0 | 4.2 | 4.1 |
| smooth | 5.6 | 4.5 | 4.5 | 3.8 | 5.3 | 4.1 | 4.2 | 4.2 |
| Pred. | house | lena | peppers | splash | tulips | us | aerial2 | bike3 |
| average | 3.9 | 3.8 | 3.6 | 3.2 | 4.0 | 3.1 | 4.0 | 3.9 |
| sharp | 3.8 | 3.9 | 3.6 | 3.3 | 4.1 | 2.8 | 4.1 | 4.0 |
| smooth | 3.9 | 3.9 | 3.6 | 3.3 | 4.1 | 2.9 | 3.9 | 4.0 |
| Pred. | bike | cafe | cats | tools | average | | | |
| average | 3.8 | 4.8 | 4.8 | 5.1 | 4.2 | | | |
| sharp | 3.8 | 4.8 | 4.8 | 5.1 | 4.2 | | | |
| smooth | 3.9 | 4.9 | 4.8 | 5.2 | 4.2 | | | |

Table D.21: Napkin for Salt and Pepper. Compressibility vs. prediction variant.

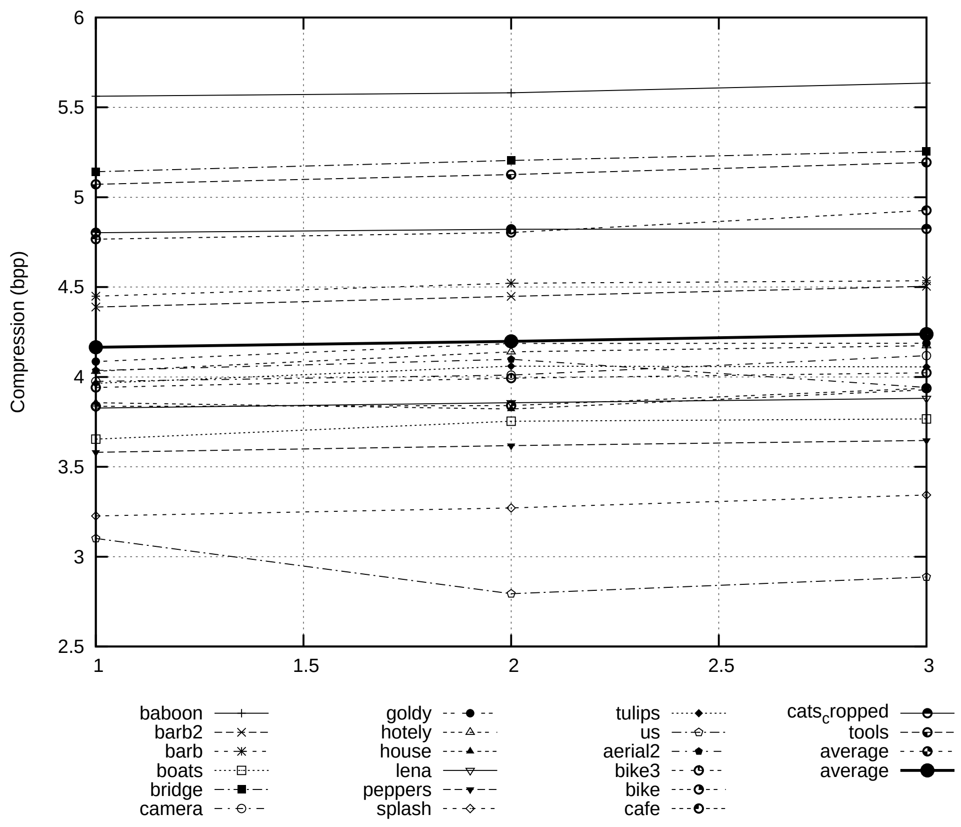


Figure D.13: Napkin for Salt and Pepper. Compressibility vs. prediction variant.

| Pred. | baboon | barb2 | barb | boats | bridge | camera | goldy | hotely |
|---|---|---|---|---|---|---|---|---|
| average | 26.4 | 29.7 | 29.4 | 35.1 | 29.1 | 29.8 | 34.8 | 33.4 |
| sharp | 26.2 | 29.5 | 29.1 | 33.9 | 28.7 | 29.5 | 33.9 | 32.5 |
| smooth | 25.8 | 28.3 | 28.7 | 33.3 | 28.1 | 28.7 | 33.6 | 31.1 |
| Pred. | house | lena | peppers | splash | tulips | us | aerial2 | bike3 |
| average | 34.9 | 35.7 | 32.6 | 39.2 | 35.1 | 27.6 | 30.9 | 29.6 |
| sharp | 33.9 | 34.6 | 31.8 | 38.0 | 33.7 | 27.6 | 30.3 | 29.3 |
| smooth | 33.1 | 34.5 | 32.0 | 38.0 | 33.5 | 26.4 | 29.6 | 28.7 |
| Pred. | bike | cafe | cats | tools | average | | | |
| average | 28.2 | 26.3 | 29.7 | 23.5 | 31.1 | | | |
| sharp | 28.0 | 26.2 | 29.0 | 23.4 | 30.5 | | | |
| smooth | 27.1 | 25.0 | 28.9 | 22.4 | 29.8 | | | |

Table D.22: Napkin for Salt and Pepper. PSNR vs. prediction variant.

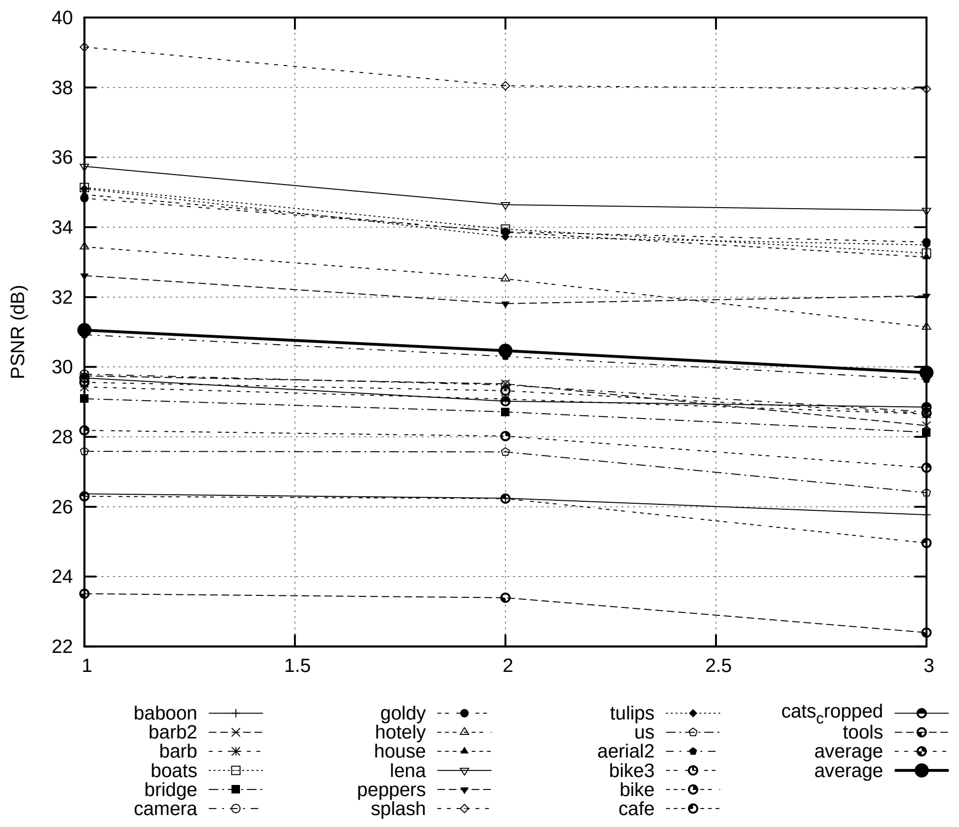


Figure D.14: Napkin for Salt and Pepper. PSNR vs. prediction variant.

| Iter. | baboon | barb2 | barb | boats | bridge | camera | goldy | hotely |
|---|---|---|---|---|---|---|---|---|
| 0 | 5.9 | 4.7 | 4.7 | 3.9 | 5.4 | 4.3 | 4.3 | 4.3 |
| 1 | 5.8 | 4.6 | 4.7 | 3.9 | 5.4 | 4.2 | 4.3 | 4.2 |
| 3 | 5.8 | 4.6 | 4.7 | 3.8 | 5.4 | 4.2 | 4.3 | 4.2 |
| 5 | 5.8 | 4.6 | 4.7 | 3.9 | 5.4 | 4.3 | 4.3 | 4.2 |
| 7 | 5.8 | 4.6 | 4.7 | 3.8 | 5.4 | 4.3 | 4.3 | 4.2 |
| 9 | 5.8 | 4.6 | 4.7 | 3.8 | 5.4 | 4.2 | 4.3 | 4.2 |
| Iter. | house | lena | peppers | splash | tulips | us | aerial2 | bike3 |
| 0 | 4.0 | 4.1 | 3.8 | 3.5 | 4.1 | 2.9 | 4.1 | 4.2 |
| 1 | 4.0 | 4.1 | 3.8 | 3.5 | 4.1 | 2.9 | 4.1 | 4.2 |
| 3 | 4.0 | 4.0 | 3.8 | 3.5 | 4.1 | 2.9 | 4.0 | 4.1 |
| 5 | 4.0 | 4.0 | 3.8 | 3.5 | 4.1 | 2.9 | 4.1 | 4.1 |
| 7 | 3.9 | 4.0 | 3.8 | 3.5 | 4.1 | 2.9 | 4.1 | 4.1 |
| 9 | 4.0 | 4.0 | 3.8 | 3.5 | 4.1 | 3.1 | 4.2 | 4.1 |
| Iter. | bike | cafe | cats | tools | average | | | |
| 0 | 4.1 | 5.0 | 5.0 | 5.3 | 4.4 | | | |
| 1 | 4.1 | 5.0 | 5.0 | 5.3 | 4.4 | | | |
| 3 | 4.1 | 5.0 | 5.0 | 5.3 | 4.3 | | | |
| 5 | 4.1 | 5.0 | 5.0 | 5.3 | 4.3 | | | |
| 7 | 4.1 | 5.0 | 5.0 | 5.3 | 4.3 | | | |
| 9 | 4.1 | 5.0 | 5.0 | 5.3 | 4.4 | | | |

Table D.23: Napkin for Salt and Pepper. Compressibility vs. iterative prefiltering applications.

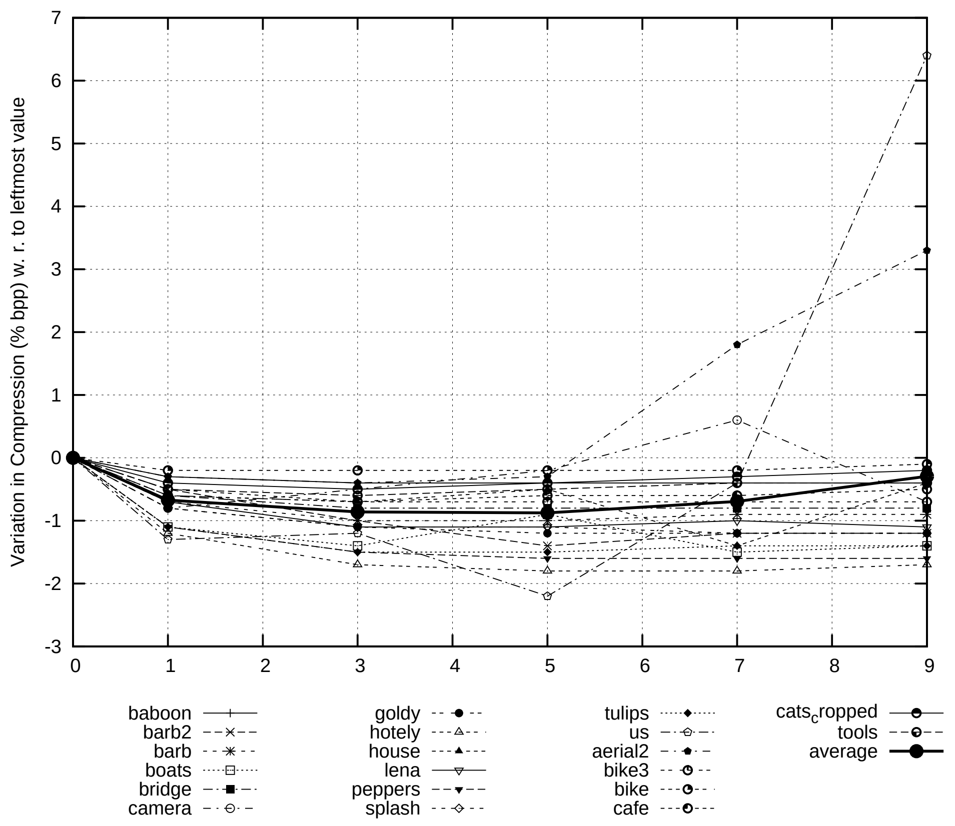


Figure D.15: Napkin for Salt and Pepper. Compressibility vs. iterative prefiltering applications.

| Iter. | baboon | barb2 | barb | boats | bridge | camera | goldy | hotely |
|---|---|---|---|---|---|---|---|---|
| 0 | 27.4 | 30.9 | 30.8 | 36.0 | 30.0 | 30.0 | 35.8 | 33.9 |
| 1 | 27.7 | 31.9 | 31.4 | 37.2 | 30.4 | 30.7 | 36.5 | 35.3 |
| 3 | 27.8 | 32.6 | 31.7 | 37.9 | 30.6 | 31.0 | 36.8 | 36.1 |
| 5 | 27.8 | 32.8 | 31.7 | 38.2 | 30.6 | 31.1 | 36.9 | 36.3 |
| 7 | 27.8 | 32.9 | 31.7 | 38.3 | 30.6 | 31.1 | 37.0 | 36.3 |
| 9 | 27.8 | 32.9 | 31.7 | 38.3 | 30.6 | 31.1 | 37.0 | 36.4 |
| Iter. | house | lena | peppers | splash | tulips | us | aerial2 | bike3 |
| 0 | 35.8 | 36.9 | 32.4 | 39.7 | 36.6 | 27.4 | 31.9 | 30.1 |
| 1 | 37.1 | 37.7 | 32.9 | 40.7 | 37.5 | 28.2 | 32.5 | 30.6 |
| 3 | 38.0 | 38.1 | 33.3 | 41.3 | 37.9 | 28.5 | 32.8 | 30.8 |
| 5 | 38.3 | 38.2 | 33.4 | 41.5 | 38.0 | 28.7 | 32.8 | 30.8 |
| 7 | 38.4 | 38.2 | 33.4 | 41.5 | 37.9 | 28.7 | 32.8 | 30.7 |
| 9 | 38.5 | 38.2 | 33.4 | 41.6 | 37.9 | 28.7 | 32.8 | 30.7 |
| Iter. | bike | cafe | cats | tools | average | | | |
| 0 | 28.4 | 26.5 | 31.3 | 23.6 | 31.8 | | | |
| 1 | 29.2 | 27.2 | 31.7 | 24.1 | 32.5 | | | |
| 3 | 29.5 | 27.5 | 31.8 | 24.4 | 32.9 | | | |
| 5 | 29.6 | 27.6 | 31.8 | 24.5 | 33.0 | | | |
| 7 | 29.6 | 27.6 | 31.8 | 24.5 | 33.0 | | | |
| 9 | 29.7 | 27.6 | 31.7 | 24.5 | 33.0 | | | |

Table D.24: Napkin for Salt and Pepper. PSNR vs. iterative prefiltering applications.

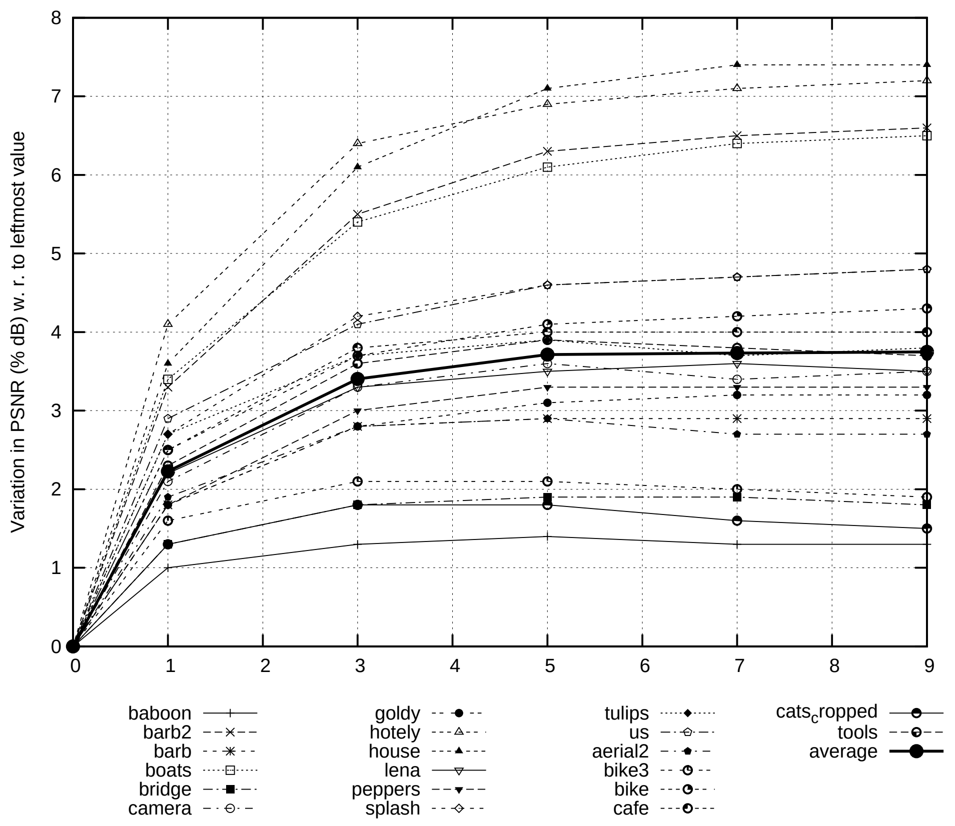


Figure D.16: Napkin for Salt and Pepper. PSNR vs. iterative prefiltering applications.

| Iter. | baboon | barb2 | barb | boats | bridge | camera | goldy | hotely |
|---|---|---|---|---|---|---|---|---|
| 1 | 5.9 | 4.7 | 4.8 | 4.2 | 5.4 | 4.5 | 4.5 | 4.4 |
| 3 | 5.9 | 4.7 | 4.8 | 4.1 | 5.4 | 4.5 | 4.5 | 4.4 |
| 5 | 5.9 | 4.7 | 4.8 | 4.1 | 5.4 | 4.5 | 4.5 | 4.4 |
| 7 | 5.9 | 4.7 | 4.8 | 4.1 | 5.4 | 4.6 | 4.5 | 4.5 |
| Iter. | house | lena | peppers | splash | tulips | us | average | |
| 1 | 4.4 | 4.3 | 4.1 | 3.8 | 4.3 | 3.5 | 4.5 | |
| 3 | 4.2 | 4.3 | 4.1 | 3.8 | 4.4 | 3.8 | 4.5 | |
| 5 | 4.2 | 4.3 | 4.0 | 3.9 | 4.4 | 4.0 | 4.5 | |
| 7 | 4.3 | 4.3 | 4.1 | 3.9 | 4.4 | 4.1 | 4.5 | |

Table D.25: Combined LBG/Napkin for Salt and Pepper. Compressibility vs. iterative prefiltering applications.

| Iter. | baboon | barb2 | barb | boats | bridge | camera | goldy | hotely |
|---|---|---|---|---|---|---|---|---|
| 1 | 27.9 | 32.9 | 33.9 | 37.2 | 30.6 | 30.2 | 36.5 | 35.8 |
| 3 | 28.1 | 33.5 | 34.6 | 38.0 | 30.7 | 30.2 | 36.8 | 36.4 |
| 5 | 28.1 | 33.7 | 34.9 | 38.2 | 30.7 | 30.3 | 36.9 | 36.5 |
| 7 | 28.1 | 33.7 | 34.9 | 38.1 | 30.7 | 30.4 | 37.0 | 36.4 |
| Iter. | house | lena | peppers | splash | tulips | us | average | |
| 1 | 34.3 | 37.6 | 34.4 | 39.6 | 37.5 | 28.5 | 34.0 | |
| 3 | 34.8 | 37.9 | 35.4 | 39.8 | 37.8 | 28.8 | 34.5 | |
| 5 | 35.4 | 37.8 | 35.8 | 39.7 | 37.7 | 27.4 | 34.5 | |
| 7 | 35.1 | 37.7 | 36.0 | 39.4 | 37.6 | 25.1 | 34.3 | |

Table D.26: Combined LBG/Napkin for Salt and Pepper. PSNR vs. iterative prefiltering applications.

### D.2.3 Combined LBG/Napkin for Salt and Pepper

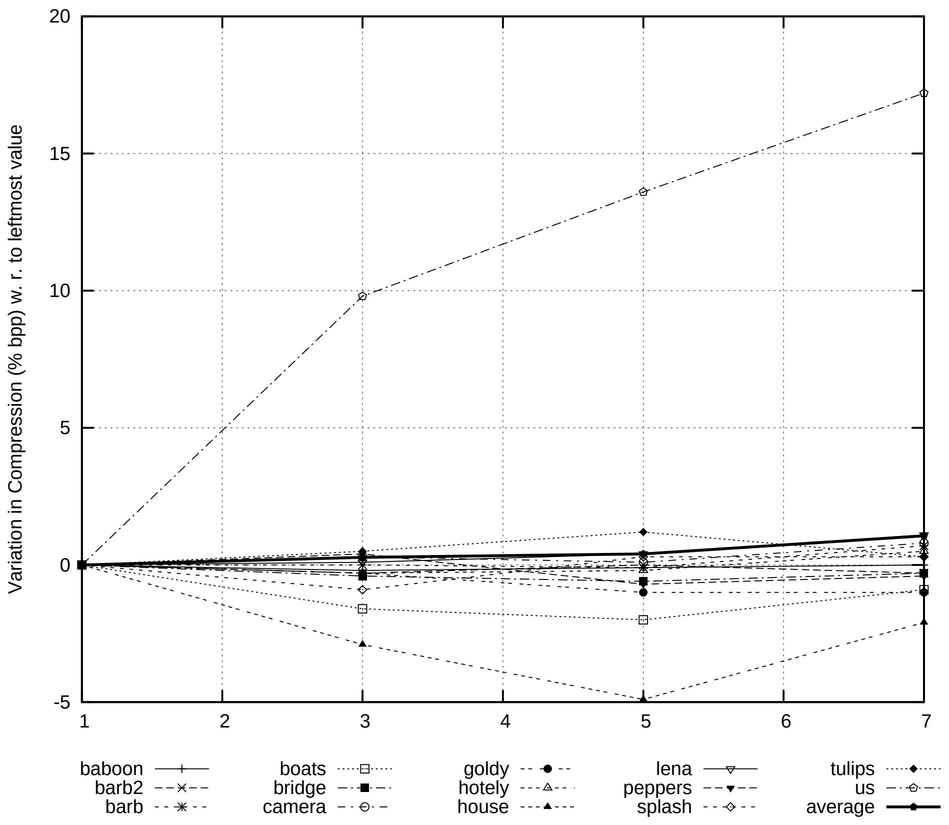


Figure D.17: Combined LBG/Napkin for Salt and Pepper. Compressibility vs. iterative pre-filtering applications.

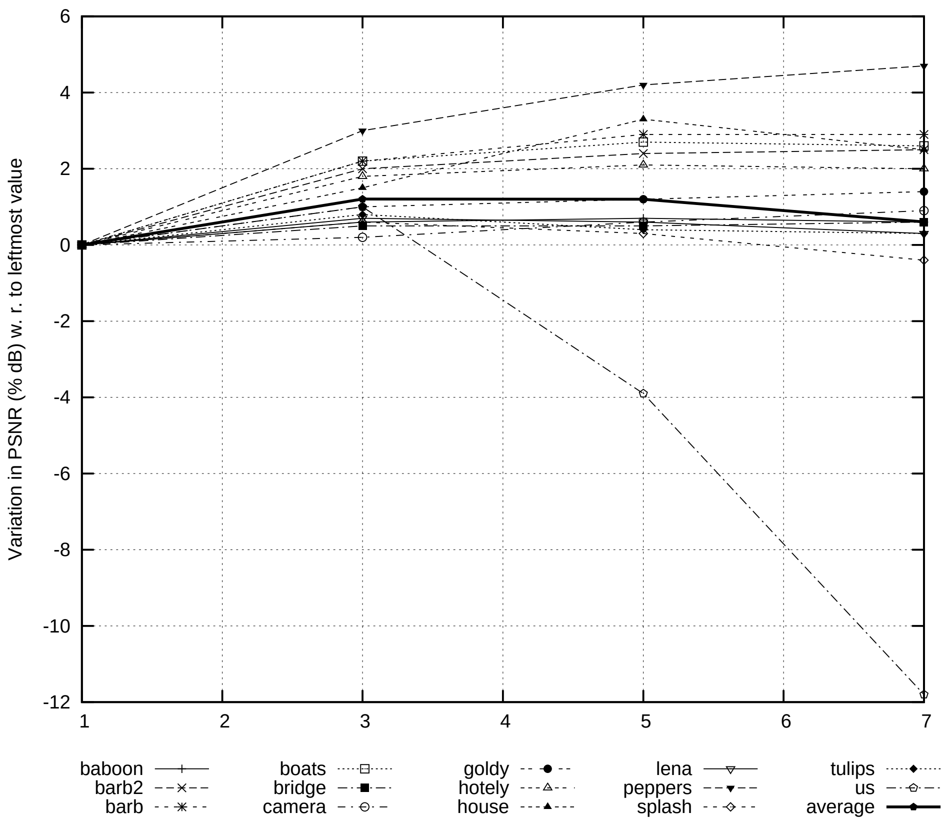


Figure D.18: Combined LBG/Napkin for Salt and Pepper. PSNR vs. iterative prefiltering applications.

| Iter. | baboon | barb2 | barb | boats | bridge | camera | goldy | hotely |
|---|---|---|---|---|---|---|---|---|
| 4 | 6.0 | 5.1 | 5.1 | 4.4 | 5.5 | 4.8 | 4.4 | 4.6 |
| 8 | 6.0 | 5.1 | 5.1 | 4.4 | 5.5 | 4.8 | 4.4 | 4.6 |
| 16 | 6.0 | 5.1 | 5.1 | 4.4 | 5.5 | 4.7 | 4.4 | 4.6 |
| 32 | 5.9 | 5.1 | 5.1 | 4.4 | 5.5 | 4.8 | 4.4 | 4.6 |
| 64 | 5.9 | 5.1 | 5.1 | 4.4 | 5.5 | 4.8 | 4.4 | 4.6 |
| Iter. | house | lena | peppers | splash | tulips | us | average | |
| 4 | 4.6 | 4.4 | 4.2 | 4.1 | 4.5 | 4.0 | 4.7 | |
| 8 | 4.6 | 4.4 | 4.2 | 4.1 | 4.5 | 4.2 | 4.7 | |
| 16 | 4.6 | 4.4 | 4.2 | 4.1 | 4.5 | 4.1 | 4.7 | |
| 32 | 4.5 | 4.4 | 4.2 | 4.1 | 4.5 | 4.2 | 4.7 | |
| 64 | 4.6 | 4.4 | 4.2 | 4.1 | 4.5 | 4.3 | 4.7 | |

Table D.27: Legacy for Gaussian noise. Comp. vs. LBG iterations.

### D.2.4 Legacy for Gaussian noise

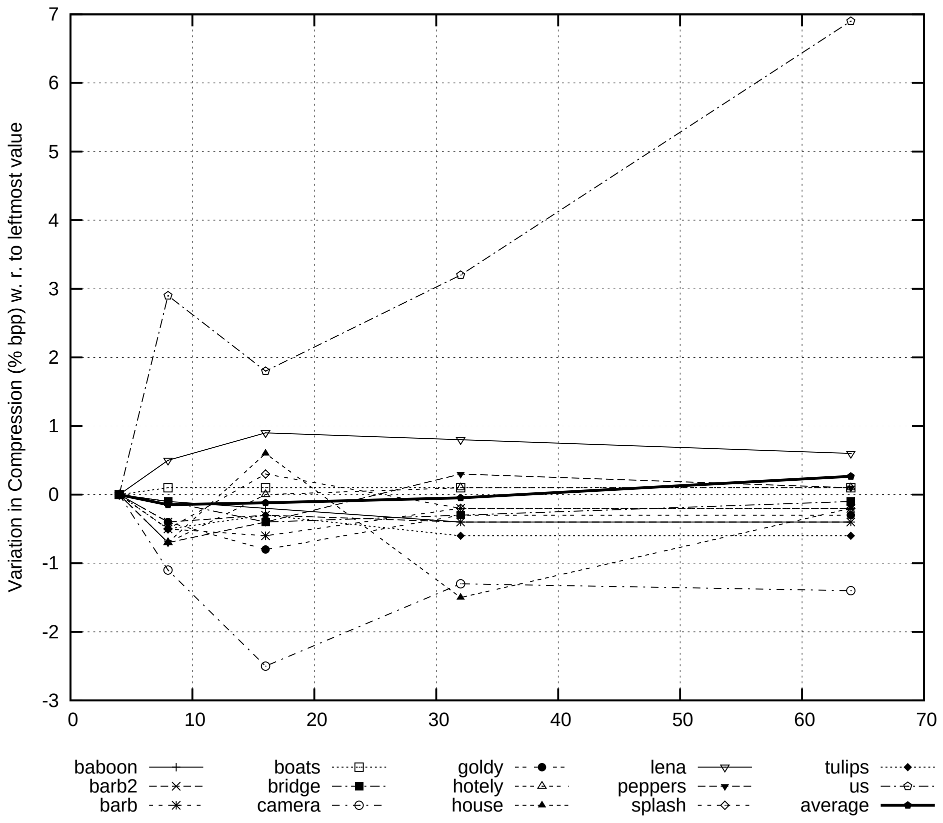


Figure D.19: Legacy for Gaussian noise. Comp. vs. LBG iterations.

| Iter. | baboon | barb2 | barb | boats | bridge | camera | goldy | hotely |
|---|---|---|---|---|---|---|---|---|
| 4 | 25.4 | 27.6 | 27.8 | 29.9 | 26.4 | 28.1 | 29.6 | 29.4 |
| 8 | 25.4 | 27.6 | 28.0 | 30.0 | 26.4 | 28.1 | 29.6 | 29.4 |
| 16 | 25.4 | 27.7 | 28.0 | 30.0 | 26.4 | 28.1 | 29.7 | 29.5 |
| 32 | 25.5 | 27.7 | 28.1 | 30.0 | 26.4 | 28.1 | 29.7 | 29.5 |
| 64 | 25.5 | 27.7 | 28.1 | 30.0 | 26.4 | 28.1 | 29.7 | 29.5 |
| Iter. | house | lena | peppers | splash | tulips | us | average | |
| 4 | 30.1 | 30.5 | 31.1 | 31.2 | 30.2 | 28.9 | 29.0 | |
| 8 | 30.2 | 30.6 | 31.2 | 31.1 | 30.2 | 29.0 | 29.1 | |
| 16 | 30.3 | 30.6 | 31.3 | 31.1 | 30.3 | 29.0 | 29.1 | |
| 32 | 30.3 | 30.6 | 31.3 | 31.1 | 30.3 | 29.1 | 29.1 | |
| 64 | 30.3 | 30.6 | 31.3 | 31.1 | 30.3 | 29.0 | 29.1 | |

Table D.28: Legacy for Gaussian noise. PSNR vs. LBG iterations.

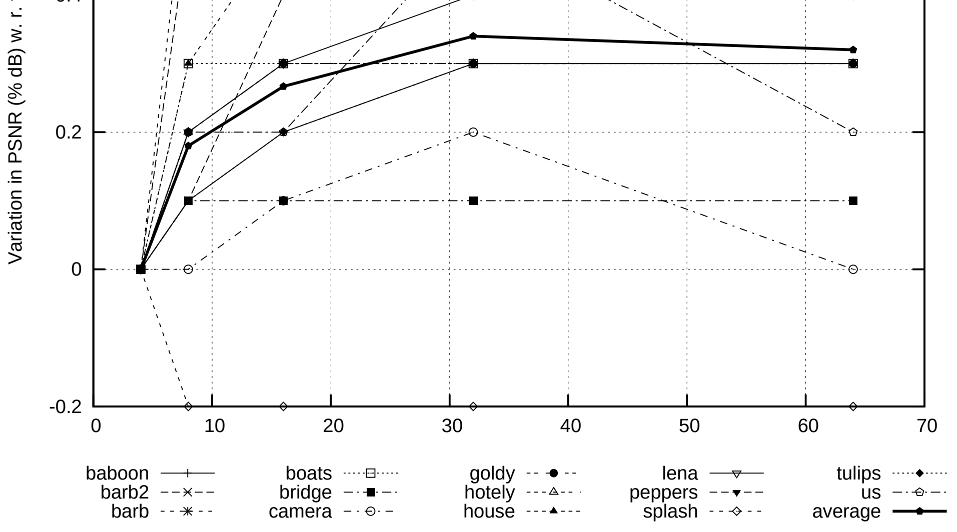


Figure D.20: Legacy for Gaussian noise. PSNR vs. LBG iterations.

| Pred. | baboon | barb2 | barb | boats | bridge | camera | goldy | hotely |
|---|---|---|---|---|---|---|---|---|
| average | 5.9 | 5.1 | 5.1 | 4.4 | 5.5 | 4.8 | 4.4 | 4.6 |
| median | 6.0 | 5.1 | 5.2 | 4.5 | 5.5 | 4.8 | 4.5 | 4.7 |
| gaussian | 5.9 | 5.0 | 5.0 | 4.3 | 5.4 | 4.7 | 4.3 | 4.5 |
| Pred. | house | lena | peppers | splash | tulips | us | average | |
| average | 4.6 | 4.4 | 4.2 | 4.1 | 4.5 | 4.3 | 4.7 | |
| median | 4.7 | 4.5 | 4.3 | 4.1 | 4.6 | 4.6 | 4.8 | |
| gaussian | 4.4 | 4.2 | 4.1 | 4.0 | 4.4 | 4.3 | 4.6 | |

Table D.29: Legacy for Gaussian noise. Compressibility vs. predictor.

| Pred. | baboon | barb2 | barb | boats | bridge | camera | goldy | hotely |
|---|---|---|---|---|---|---|---|---|
| average | 25.5 | 27.7 | 28.0 | 30.0 | 26.4 | 28.1 | 29.7 | 29.5 |
| median | 25.4 | 27.7 | 27.9 | 29.8 | 26.3 | 27.9 | 29.5 | 29.2 |
| gaussian | 25.5 | 27.8 | 28.2 | 30.2 | 26.5 | 28.2 | 29.8 | 29.6 |
| Pred. | house | lena | peppers | splash | tulips | us | average | |
| average | 30.3 | 30.6 | 31.3 | 31.1 | 30.3 | 29.0 | 29.1 | |
| median | 30.0 | 30.4 | 31.0 | 32.3 | 30.1 | 26.4 | 28.8 | |
| gaussian | 30.4 | 30.8 | 31.4 | 31.4 | 30.5 | 29.2 | 29.3 | |

Table D.30: Legacy for Gaussian noise. PSNR vs. predictor.

| Iter. | baboon | barb2 | barb | boats | bridge | camera | goldy | hotely |
|---|---|---|---|---|---|---|---|---|
| 0 | 5.9 | 5.1 | 5.1 | 4.4 | 5.5 | 4.8 | 4.4 | 4.6 |
| 1 | 5.7 | 5.0 | 4.8 | 4.2 | 5.4 | 4.5 | 4.3 | 4.3 |
| 2 | 5.7 | 5.1 | 5.0 | 4.6 | 5.4 | 4.9 | 4.7 | 4.7 |
| 4 | 5.7 | 5.1 | 4.9 | 4.4 | 5.3 | 4.7 | 4.4 | 4.5 |
| Iter. | house | lena | peppers | splash | tulips | us | average | |
| 0 | 4.6 | 4.4 | 4.2 | 4.1 | 4.5 | 4.3 | 4.7 | |
| 1 | 4.1 | 4.0 | 3.9 | 4.0 | 4.4 | 3.6 | 4.4 | |
| 2 | 4.3 | 4.5 | 4.4 | 4.3 | 4.6 | 4.3 | 4.7 | |
| 4 | 4.5 | 4.3 | 4.2 | 4.2 | 4.4 | 3.8 | 4.6 | |

Table D.31: Legacy for Gaussian noise. Compressibility vs. iterative prefiltering applications.

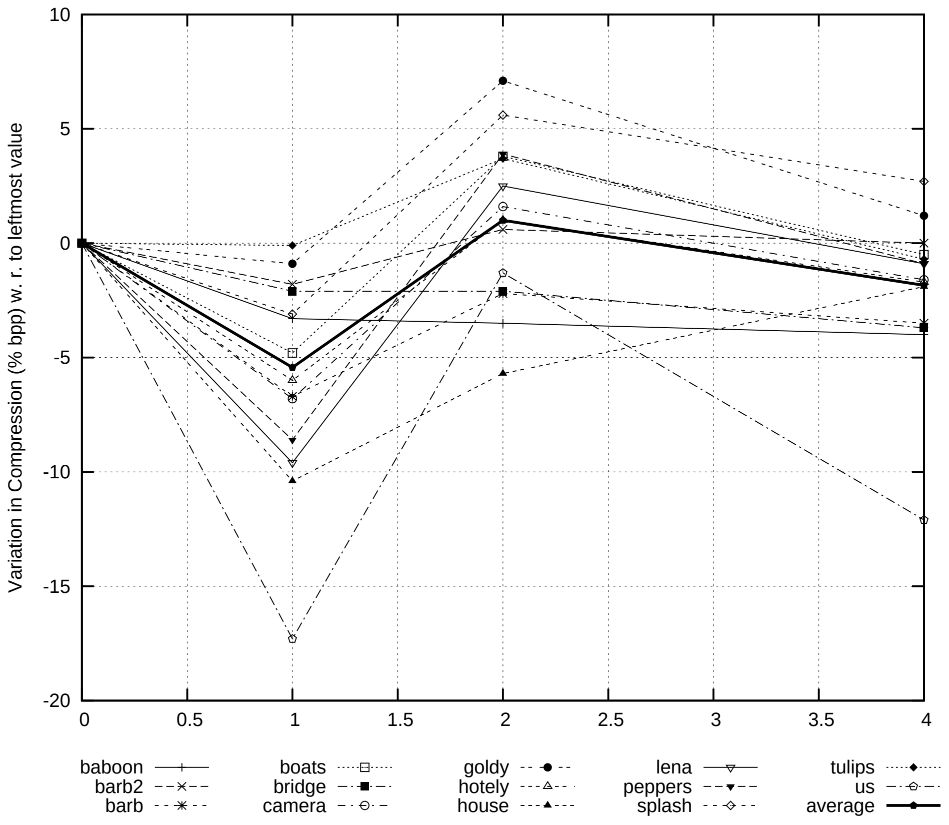


Figure D.21: Legacy for Gaussian noise. Compressibility vs. iterative prefiltering applications.

| Iter | baboon | barb2 | barb | boats | bridge | camera | goldy | hotely |
|---|---|---|---|---|---|---|---|---|
| 0.00 | 25.5 | 27.7 | 28.0 | 30.0 | 26.4 | 28.1 | 29.7 | 29.5 |
| 1.00 | 25.6 | 27.6 | 28.3 | 30.3 | 26.4 | 28.5 | 29.7 | 29.7 |
| 2.00 | 25.6 | 27.3 | 28.3 | 29.9 | 26.4 | 28.2 | 29.1 | 29.3 |
| 4.00 | 25.6 | 27.2 | 28.1 | 29.8 | 26.4 | 28.1 | 29.2 | 29.1 |
| Iter. | house | lena | peppers | splash | tulips | us | average | |
| 0 | 30.3 | 30.6 | 31.3 | 31.1 | 30.3 | 29.0 | 29.1 | |
| 1 | 31.0 | 31.0 | 31.7 | 31.5 | 30.3 | 29.3 | 29.3 | |
| 2 | 30.8 | 30.3 | 30.8 | 30.8 | 29.9 | 29.1 | 29.0 | |
| 4 | 30.4 | 30.2 | 30.7 | 30.2 | 29.8 | 29.1 | 28.8 | |

Table D.32: Legacy for Gaussian noise. PSNR vs. iterative prefiltering applications.

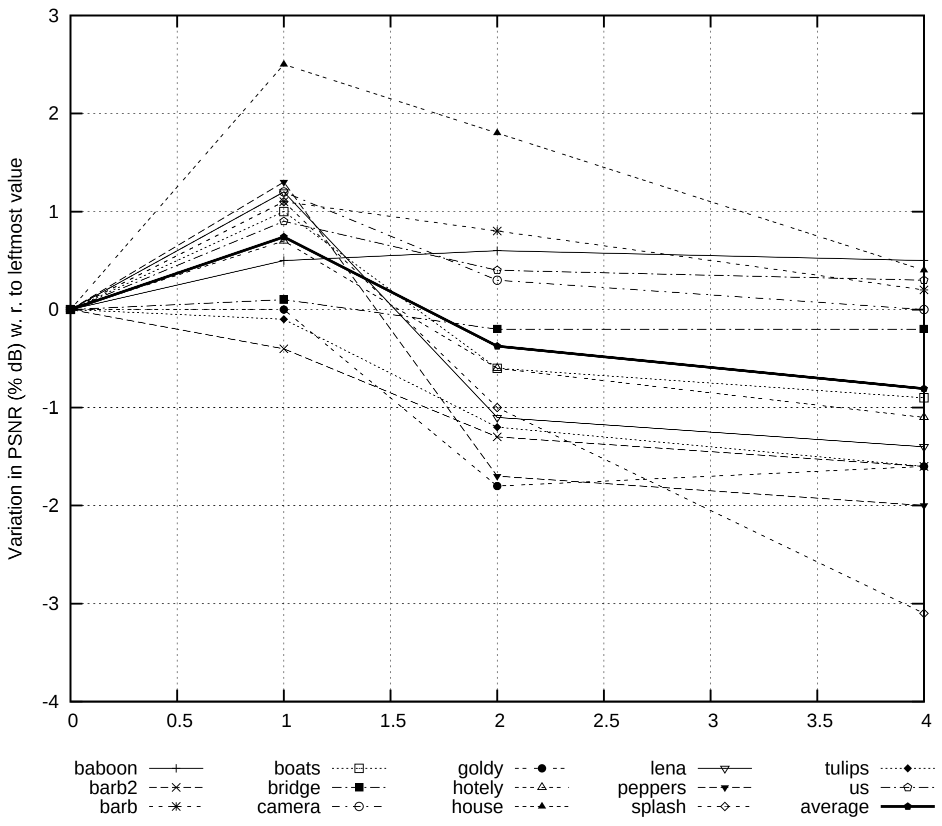


Figure D.22: Legacy for Gaussian noise. PSNR vs. iterative prefiltering applications.

| Clusters | baboon | barb2 | barb | boats | bridge | camera | goldy | hotely |
|---|---|---|---|---|---|---|---|---|
| 32 | 6.0 | 5.1 | 5.2 | 4.4 | 5.5 | 4.7 | 4.4 | 4.7 |
| 64 | 6.0 | 5.1 | 5.1 | 4.4 | 5.5 | 4.7 | 4.4 | 4.6 |
| 96 | 6.0 | 5.1 | 5.1 | 4.4 | 5.5 | 4.8 | 4.4 | 4.6 |
| 128 | 5.9 | 5.1 | 5.1 | 4.4 | 5.5 | 4.8 | 4.4 | 4.6 |
| 192 | 5.9 | 5.3 | 5.1 | 4.4 | 5.5 | 4.8 | 4.4 | 4.6 |
| 256 | 5.9 | 5.5 | 5.1 | 4.4 | 5.5 | 4.9 | 4.5 | 4.6 |
| 288 | 5.9 | 5.4 | 5.1 | 4.4 | 5.5 | 4.9 | 4.4 | 4.6 |
| Clusters | house | lena | peppers | splash | tulips | us | average | |
| 32 | 4.4 | 4.3 | 4.2 | 4.0 | 4.5 | 4.0 | 4.7 | |
| 64 | 4.5 | 4.3 | 4.1 | 4.0 | 4.4 | 4.2 | 4.7 | |
| 96 | 4.5 | 4.3 | 4.2 | 4.1 | 4.5 | 4.3 | 4.7 | |
| 128 | 4.6 | 4.4 | 4.2 | 4.1 | 4.5 | 4.3 | 4.7 | |
| 192 | 4.6 | 4.4 | 4.2 | 4.2 | 4.5 | 4.2 | 4.7 | |
| 256 | 4.7 | 4.4 | 4.3 | 4.4 | 4.5 | 4.2 | 4.8 | |
| 288 | 4.8 | 4.4 | 4.3 | 4.4 | 4.5 | 4.2 | 4.8 | |

Table D.33: Legacy for Gaussian noise. Compressibility vs number of context clusters.

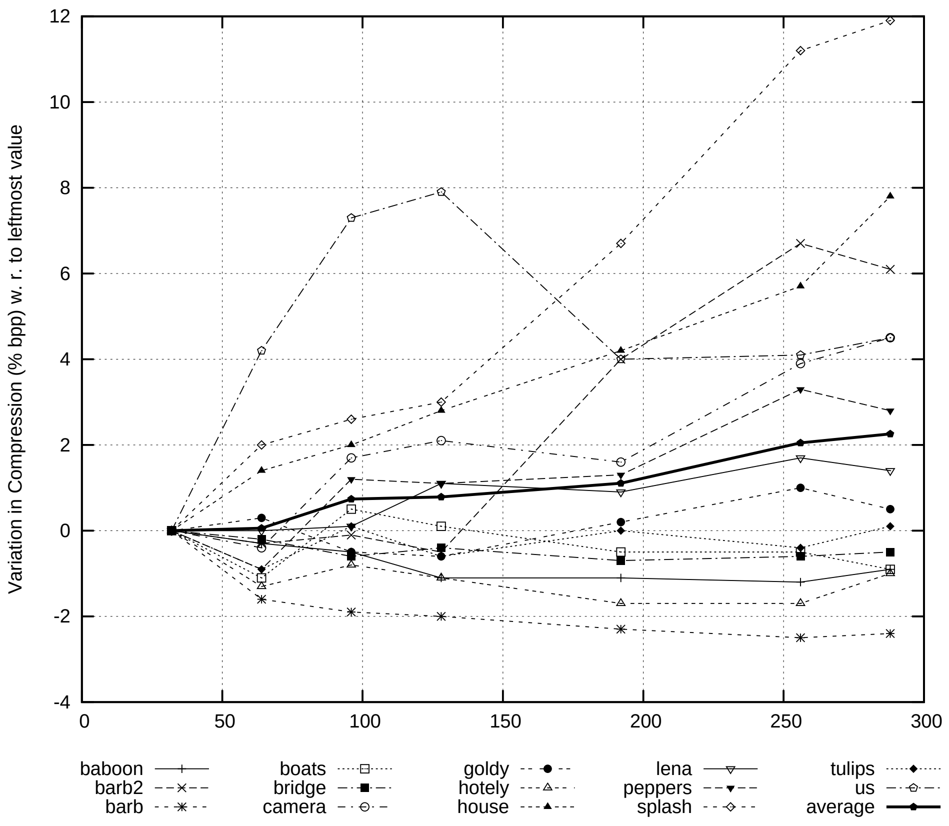


Figure D.23: Legacy for Gaussian noise. Compressibility vs number of context clusters.

| Clusters | baboon | barb2 | barb | boats | bridge | camera | goldy | hotely |
|---|---|---|---|---|---|---|---|---|
| 32.00 | 25.3 | 27.4 | 27.3 | 29.8 | 26.3 | 27.9 | 29.6 | 29.1 |
| 64.00 | 25.4 | 27.5 | 27.8 | 30.0 | 26.4 | 28.0 | 29.6 | 29.3 |
| 96.00 | 25.4 | 27.6 | 28.0 | 30.1 | 26.4 | 28.1 | 29.7 | 29.4 |
| 128.00 | 25.5 | 27.7 | 28.0 | 30.0 | 26.4 | 28.1 | 29.7 | 29.5 |
| 192.00 | 25.5 | 27.0 | 28.1 | 30.2 | 26.4 | 28.1 | 29.7 | 29.5 |
| 256.00 | 25.5 | 26.7 | 28.3 | 30.2 | 26.4 | 28.1 | 29.7 | 29.6 |
| 288.00 | 25.5 | 26.7 | 28.3 | 30.2 | 26.4 | 28.1 | 29.7 | 29.6 |
| Clusters | house | lena | peppers | splash | tulips | us | average | |
| 32 | 30.1 | 30.5 | 31.1 | 31.1 | 30.0 | 28.5 | 28.9 | |
| 64 | 30.3 | 30.6 | 31.2 | 31.1 | 30.2 | 28.9 | 29.0 | |
| 96 | 30.3 | 30.6 | 31.2 | 31.1 | 30.2 | 28.9 | 29.1 | |
| 128 | 30.3 | 30.6 | 31.3 | 31.1 | 30.3 | 29.0 | 29.1 | |
| 192 | 30.3 | 30.6 | 31.2 | 30.8 | 30.3 | 29.3 | 29.1 | |
| 256 | 30.2 | 30.6 | 30.8 | 29.9 | 30.4 | 29.3 | 29.0 | |
| 288 | 30.1 | 30.6 | 30.9 | 29.7 | 30.4 | 29.5 | 29.0 | |

Table D.34: Legacy for Gaussian noise. PSNR vs number of context clusters.

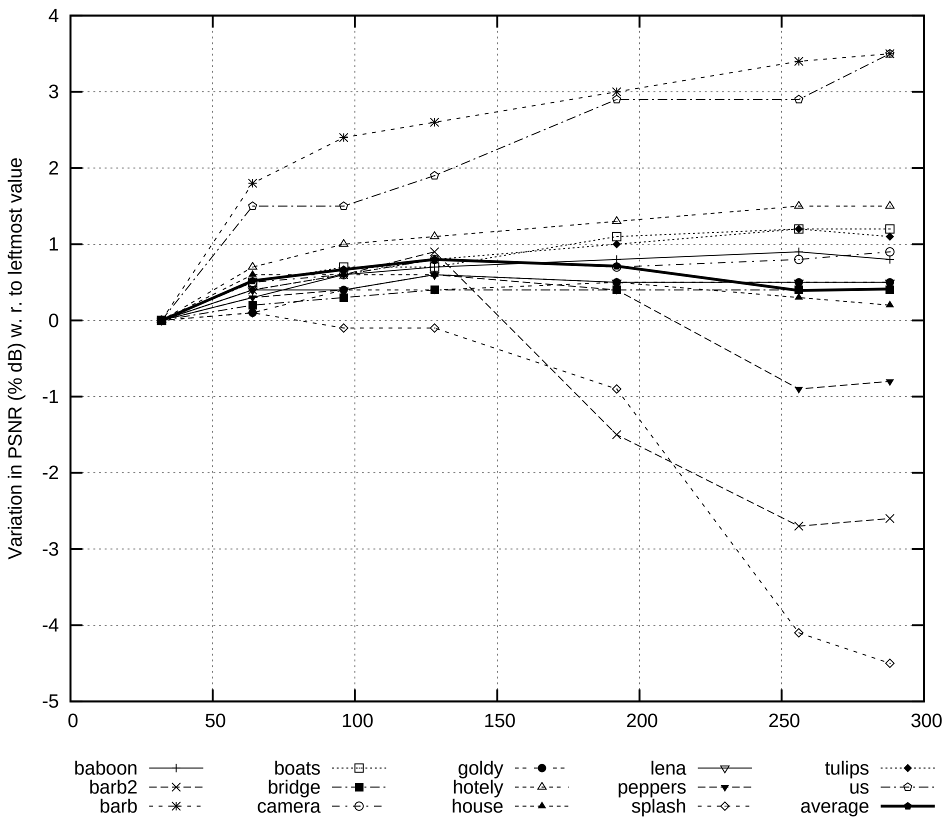


Figure D.24: Legacy for Gaussian noise. PSNR vs number of context clusters.

# Bibliography

[1] Antoni Buades, Bartomeu Coll, and Jean-Michel Morel. A non-local algorithm for image denoising. In *Proceedings of CVPR (2)*, pages 60–65, 2005.

[2] Raymond H. Chan, Chung-Wa Ho, and Mila Nikolova. Salt-and-pepper noise removal by median-type noise detectors and edge-preserving regularization. *IEEE Transactions on Image Processing*, 14:1479–1485, 2005.

[3] R. R. Coiffman and David L. Donoho. Translation invariant de-noising. In *Wavelets and Statistics*, pages 125–150. Springer-Verlag, 1995.

[4] Thomas Cover and Joy Thomas. *Elements of Information Theory.* Wiley-Interscience, 1 edition, 1991.

[5] Ingrid Daubechies. *Ten Lectures on Wavelets.* CBMS-NSF Reg. Conf. Series in Applied Math. SIAM, 1992.

[6] David L. Donoho and Iain M. Johnstone. Ideal spatial adaptation by wavelet shrinkage. *Biometrika*, 81:425–455, 1994.

[7] Peter O. Duda and Hart. *Pattern Recognition.* Wiley-Interscience, 2 edition, 2001.

[8] Gene H. Golub and Charles F. Van Loan. *Matrix Computations.* Johns Hopkins University Press, 3 edition, 1996.

[9] Rafael Gonzalez and Paul Wintz. *Digital Image Processing.* Addison-Wesley, 2 edition, 1987.

[10] R. Haralick and L. Shapiro. *Computer and Robot Vision*, volume 1, chapter 7. Addison-Wesley, 1992.

[11] Monson H. Hayes. *Statistical Signal Processing and Modeling.* Wiley, 1996.

[12] H. Hwang and R. Haddad. Adaptive median filters: new algorithms and results. In *Proceedings of ICIP*, volume 4, pages 499–502. IEEE, 1995.

[13] R. E. Kalman. A new approach to linear filtering and prediction problems. *Basic Engeneering*, 82:34–45, 1960.

[14] J. S. Lee. Digital image enhancement and noise filtering by use of local statistics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2:165–168, 1980.

[15] Y. Linde, A. Buzo, and R. Gray. An algorithm for vector quantizer design. *IEEE Transactions on Communications*, 28:84–94, 1980.

[16] Stéphane Mallat. *A Wavelet Tour of Signal Processing.* Academic Press, 1 edition, 1998.

[17] Neri Merhav, Gadiel Seroussi, and Marcelo Weinberger. Lossless compression for sources with two-sided geometric distributions. Technical report, Hewlett-Packard Laboratories Palo Alto, 1998.

[18] Giovanni Motta. Discrete universal denoiser: Adaptation issues for graylevel images. Technical report, Hewlett-Packard Laboratories Palo Alto, March 2003.

[19] J. O'Brien. Predictive quantizing differential pulse code modulation for the transmission of television signals. *Bell Systems Tech Journal*, 45:689–722, May 1966.

[20] Alan V. Oppenheim and Ronald W. Schafer. *Digital Signal Processing.* Prentice Hall, 1975.

[21] Erik Ordentlich, Gadiel Seroussi, Sergio Verdú, Marcelo J. Weinberger, and Tsachy Weissman. A discrete universal denoiser and its application to binary images. In *Proceedings of ICIP*, pages 117–120, 2003.

[22] Erik Ordentlich, Marcelo J. Weinberger, and Tsachy Weissman. Multi-directional context sets with applications to universal denoising and compression. In *Proceedings of IEEE International Symposium on Information Theory*, pages 1270–1274, 2005 September.

[23] P. Perona and J. Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12:629–639, 1990.

[24] Gouchol Pok, Jyh-Charn Liu, and Attoor Sanju Nair. Selective removal of impulse noise based on homogeneity level information. *IEEE Transactions on Image Processing*, 12:85–92, January 2003.

[25] J. Portilla, V. Strela, M. J. Wainwright, and E. P. Simoncelli. Image denoising using scale mixtures of gaussians in the wavelet domain. *IEEE Transactions on Image Processing*, 12(11):1338–1351, November 2003.

[26] Jorma Rissanen. Universal coding, information, prediction, and estimation. *IEEE Transactions on Information Theory*, IT-30:629–636, July 1984.

[27] Stefan Roth and Michael J. Black. Fields of experts: A framework for learning image priors. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 860–867, June 2005.

[28] Leonid I. Rudin and Stanley Osher. Total variation based image restoration with free local constraints. In *Proceedings of ICIP*, pages 31–35, 1994.

[29] G. Sapiro. *Geometric Partial Differential Equations and Image Analysis.* Cambridge University Press, January 2001.

[30] J. L. Stark, E. J. Candès, and D. L. Donoho. Very high quality image restoration by combining wavelets and curvelets. In *Proceedings of SPIE*, 4478, 2001. Wavelet Applications in Signal and Image Processing IX.

[31] David Tschumperlé. LIC-based regularization of multi-valued images. In *Proceedings of ICIP*, 1168, 2005.

[32] M. J. Weinberger and G. Seroussi. Sequential prediction and ranking in universal context modeling and data compression. *IEEE Transactions on Information Theory*, 43:1697–1706, September 1997.

[33] M. J. Weinberger, G. Seroussi, and G. Sapiro. The LOCO-I lossless image compression algorithm: Principles and standarization into JPEG-LS. *IEEE Transactions on Image Processing*, 9:1309–1324, August 2000.

[34] Tsachy Weissman, Erik Ordentlich, Gadiel Seroussi, Sergio Verdú, and Marcelo J. Weinberger. Universal discrete denoising: known channel. *IEEE Transactions on Information Theory*, 51(1):5–28, 2005.

[35] Norbert Wiener. *Extrapolation, Interpolation, and Smoothing of Stationary Time Series.* Wiley, 1949.

[36] X. Wu and N. D. Memon. Context-based, adaptive, lossless image coding. *IEEE Transactions on Communications*, 45:437–444, April 1997.