UNIVERSIDADE DO ESTADO DO RIO GRANDE DO NORTE
UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Nícholas André P. de Oliveira

Single image super-resolution method based on linear regression and Box-Cox transformation

Mossoró-RN

2018

Nícholas André P. de Oliveira

Single image super-resolution method based on linear regression and Box-Cox transformation

Thesis submitted in partial fulfillment of the requirements for the degree of Master of Computer Science in the *Programa de Pós-Graduação em Ciência da Computação* at UERN/UFERSA.

Thesis Supervisor: Prof. Dra. Angélica Félix de Castro
Thesis Co-Supervisor: Prof. Dr. Leandro Carlos de Souza

Mossoró-RN

2018

O48s Oliveira, Nícholas André Pinho.
Single Image Super Resolution method based on LinearRegression and Box-Cox transform / Nícholas André Pinho Oliveira. - 2018.
74 f. : il.

Orientadora: Angélica Félix de Castro.
Coorientador: Leandro Carlos de Souza.
Dissertação (Mestrado) - Universidade Federal Rural do Semi-árido, Programa de Pós-graduação em Ciência da Computação, 2018.

1. digital image processing. 2. super-resolution. 3. linear regression. 4. box-cox. 5. image zooming. I. Castro, Angélica Félix de, orient. II. Souza, Leandro Carlos de, co-orient. III. Título.

NÍCHOLAS ANDRÉ PINHO DE OLIVEIRA

SINGLE IMAGE SUPER-RESOLUTION METHOD BASED ON LINEAR REGRESSION AND BOX-COX TRANSFORMATION

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação para a obtenção do título de Mestre em Ciência da Computação.

APROVADA EM: 05 / 12 / 2018

Angélica Félix de Castro
Profa. Dra. Angélica Félix de Castro
(Orientadora – UFERSA)

Leandro C. Souza
Prof. Dr. Leandro Carlos de Souza
(Coorientador – UFERSA)

[illegible]
Prof. Dr. Leiva Casemiro Oliveira
(Examinador Interno – UFERSA)

[illegible]
Prof. Dr. Daniel Faustino Lacerda de Souza
(Examinador Interno – UFERSA)

[illegible]
Profa. Dra. Renata Maria Cardoso Rodrigues de Souza
(Examinadora Externa – UFPE)

"A picture is worth a thousand words."

(John McCarthy)

Abstract

Image super-resolution is an essential operation in digital image processing and aims at recovering a high-resolution image from one or more low-resolution input images. The process in which a high-resolution image is obtained involves the creation of new and unknown pixels that must be estimated. Super-resolution (SR) algorithms can be classified as multi-image SR and single-image SR. Multi-image SR methods utilize multiple low-resolution image as input, whereas single-image SR methods utilize only one low-resolution image. Traditional methods such as nearest interpolation, bicubic interpolation and bilinear interpolation are all example of single-image SR methods. These methods, while simple, introduces many artifacts in the high-resolution images. Several methods have emerged in the past decade with the goal of obtaining better, artifacts-free high-resolution images. This thesis proposes a novel single image super-resolution method called BCZ(Box-Cox Zoom), that uses a multiple linear regression model and the Box-Cox transform to interpolate the unknown pixels in the high-resolution image. Our proposed method is compared against the classical interpolation methods using the Ultra Eye image data set. The methods are compared by building confidence intervals using the bootstrap method.

Keywords: digital image processing, super-resolution, box-cox, image zooming.

Resumo

Super-Resolução de imagens é uma operação essencial no processamento digital de imagens e objetiva recuperar uma imagem de alta resolução a partir de uma ou mais imagens de baixa resolução. O processo pelo qual a imagem de alta resolução é obtida envolve de criação de novos e desconhecidos pontos (pixels) que necessitam ser estimados. Algoritmos de Super-Resolução (SR) podem ser classificados como múltiplas imagens ou de imagem única. Métodos de múltiplas imagens utilizam diversas imagens de baixa resolução como entrada, enquanto que métodos de imagem única utiliza somente uma imagem de baixa resolução como entrada. Métodos tradicionais, tais como interpolação por replicação, interpolação bicúbica e interpolação bilinear são todos exemplos de métodos SR de imagem única. Estes métodos simples, introduzem diversos artefatos nas imagens de alta resolução resultantes. Vários outros métodos emergiram na década passada com o objetivo de obter imagens melhores e livres de artefatos. Esta dissertação propõe um novo algoritmo de super-resolução de imagem única chamado BCZ(Box-Cox Zoom), que utiliza regressão linear múltipla e a transformada de Box-Cox para interpolar os pixels desconhecidos na imagem resultante. O método proposto é comparado com os métodos de interpolação clássicos utilizando o banco de imagens Ultra Eye. Os métodos são comparados através da construção de intervalos de confiança usando o método boostrap.

Palavras-Chaves: processamento digital de imagens, super resolução, box-cox, ampliação de imagens.

List of Figures

List of Tables

List of Listings

List of abbreviations and acronyms

MSE	*Mean Squared Error*
PSNR	*Peak signal-to-noise ratio*
LSE	*Least Square Estimation*
SSIM	*Structural Similarity Index*
HR	*High Resolution*
LR	*Low Resolution*
SR	*Super-Resolution*
HVS	*Human Visual System*

Contents

1 Introduction

1.1 Overview

Image zooming is a well-known and ubiquitous image processing problem. Image zooming refers to the task of increasing or reducing the number of pixels in a given image and therein changing the spatial resolution of a digital image. Reducing the number of pixels is known as *zoom out* and is generally a much simpler problem to solve. In the other hand, increasing the resolution of a image, known as *zoom in* is a more complicated problem. Such problems have also been recently referred to as *super resolution* (SR), where the overall goal is to recover a high resolution (HR) image from one or more low resolution (LR) input images (GONZALEZ; WOODS, 2006).

A digital image is essentially a two-dimensional array (per color channel, or a single two-dimensional array for grayscale images) containing color intensity values (FLORIN; ARSINTE; MASTORAKIS, 2011a). Increasing the resolution of an image introduces new and unknown pixels in the two-dimensional array that holds the image data. Obtaining estimated values for these unknown pixels is the core of super-resolution problems. Within the scope of two-dimensional images, SR problems can be grouped into two major categories: multi-image super-resolution methods and single-image super-resolution methods.

Multi-image SR methods utilize, as the name suggests, multiple non-redundant images from the same scene (taken at subpixel misalignment) to reconstruct a HR version of the image, where each LR image imposes several linear constraints on the unknown pixels values, making it possible to solve the set of linear equations, assuming enough low input images are provided. These methods, however, are mostly numerically limited to small increases in resolution (BAKER; KANADE, 2002). Conversely, single-image SR methods have only one low input image at disposal which leads to a under-constrained and ill-posed problem. Nonetheless, in (GLASNER; BAGON; IRANI, 2009) and (YANG; LIN; COHEN, 2013) state-of-the art single-image SR methods are presented, which have proven to be superior to traditional multi-image SR methods.

Single image SR methods can be broadly classified as interpolation-based and exampled-based. Most of the traditional methods including bilinear and bicubic zoom are interpolation-based (GONZALEZ; WOODS, 2006). The majority of the interpolation methods however, do not perform well as the zooming factor increases, producing many artifacts on the resulting images such as blurred edges, ringing and aliasing artifacts. Example-based approaches requires a external database of low and high resolution images that is used to train the method. In the training step, the algorithm tries to learn the correspondence of image patches between the low and high resolution images (FREEMAN; PASZTOR, 1999).

Image super-resolution can also be classified as adaptive and non-adaptive (GONZALEZ; WOODS, 2006). Non-adaptive methods treats all pixels in the same way while adaptive methods might change the strategy utilized based on the pixel itself or its neighboring pixels. A “smart” algorithm might detect edges and apply a different strategy to estimate the pixels around the edges to avoid, for example, producing blurred edges.

The quality of the reconstructed high-resolution images ultimately depends on the strategy used to estimate the high-resolution image pixels. Traditional methods like the interpolation-based ones previously mentioned, produces several artifacts, which reduces the overall quality of the reconstructed image. These issues are usually caused by the loss of high frequency information during the zooming process. Therefore, a good SR algorithm must preserve the high frequencies of the image as much as possible as high frequencies are strongly responsible for the sharpness of the images. Measuring these artifacts, however, is not a easy task.

In order to compare the efficiency and quality of image SR methods, it becomes necessary to define models and create metrics that conveys the quality of images. Assessing digital image quality, however, is inherently a difficult task as it is a subjective assessment. Traditional metrics to evaluate image quality such as mean squared error (MSE) and peak signal-to-noise ratio (PSNR) do not take into account the perceived visual quality of images. To overcome such limitations, new methods have emerged aiming to better translate the perceived visual quality of images into a numeric value. An example of that is the structural similarity index (SSIM), a image quality framework that seeks to correlate the perceived visual difference between images with a mathematical model that combines three components, the luminance distortion, loss of correlation and the contrast distortion.

Linear Regression is a mathematical model from regression analysis that aims at obtaining a linear relationship between a response variable Y and a predictor variable X (MONTGOMERY; PECK; VINING, 2012). The most common model of regression analysis is the simple linear regression model, which is suitable for finding a relationship between one predictor variable X, i.e, with only one dimension. Multiple Linear Regression, which is an generalization of the linear model, is used when the predictor variable X has more than one dimension. There is no guarantee that the observed data is disposed linearly, therefore several transformation functions can be applied to transform the data into a more linear distribution. Box-Cox is a simple, yet powerful one. The Box-Cox transformation consists on the representation of a family of functions, where each family of functions is determined by a λ parameter (SAKIA, 1992). The λ parameter is a parameter to be determined as part of the execution of the regression model. Obtaining a good value of λ is crucial for a successful estimation. The method of maximum likelihood (RAWLINGS;

PANTULA; DICKEY, 2001) is commonly used to obtain an estimate for the λ parameter, however it is not suitable for all applications.

1.2 Contributions

This thesis proposes the BCZ (Box-Cox Zoom) method, which uses multiple linear regression and the Box-Cox transformation as an interpolation function with the goal of producing sharper images while minimizing other artifacts commonly introduced by traditional interpolation methods. The proposed method uses the same linear model for the entire image, however the interpolation function is non-linear and the algorithm is adaptive due to the fact that the Box-Cox transform is applied with different lambda values across the image. The value of the lambda parameter is determined for each image path based on the pixels distribution.

As part of this thesis, a new approach for obtaining a good estimate for the lambda parameter of the Box-Cox transformation is also proposed. In this new approach, a mathematical formula finds the lambda that minimizes the square of the residual errors. This approach proved to be more suitable for images compared to the maximum likehood method.

1.3 Methodology

To evaluate the performance of the proposed method, some widely used methods are selected to be compared against. The methodology for the evaluation and comparison of the proposed method consists of applying each of the methods being compared in a pre-selected set of test images that best represents natural images. These images have a minimum resolution of 3840x2160 pixels. Each image is broken down into several blocks of the same size which are then reduced by a factor m_i. Subsequently, these blocks are magnified by the same factor m_i. The PSNR and SSIM between the original image block and the reconstructed one is then taken and stored. The stored PSNR and SSIM values are used to build confidence intervals using the bootstrap method to provide a more reliable statistical comparison between the proposed method and the other selected methods for each one of the tested images.

1.4 Outline

This thesis is organized as outlined bellow:

Chapter 2 - Introduction to Super Image Resolution
This chapter introduces the basic concepts around image super-resolution, how they work

and the challenges of such methods. It also discuss several super-resolution methods.

Chapter 3 - Linear Regression
A brief introduction to the concepts of both simple and multiple linear regression as well as the Box-Cox transform.

Chapter 4 - Proposed Method
This Chapter is dedicated to presenting the BCZ method where we show the algorithm internals and how the Box-Cox transform was introduced to produce much better results. We also discuss the challenges faced regarding obtaining a good value for λ and how we ultimately solved that problem.

Chapter 5 - Experimental Evaluation
The experiments are reported in this chapter. The results are presented by comparing our novel method with many other methods found in the literature. Image quality assessment metrics are used to drive the comparisons. Subjective evaluation highlighting where the proposed algorithm performs particularly well is also presented.

Conclusion and Future Work
Final remarks and potential future work are outlined here.

2 Introduction to Super Image Resolution

This chapter describes the theoretical foundation behind the proposed method. The basic concepts around digital image processing, super resolution and its classes are covered in this chapter. Interpolation-based methods are presented in extreme detail to lay out a good foundation for understanding our novel image zooming method. We start with a discussion about how digital images are acquired, represented and processed inside a digital computer. Several super-resolution methods are also discussed in this chapter.

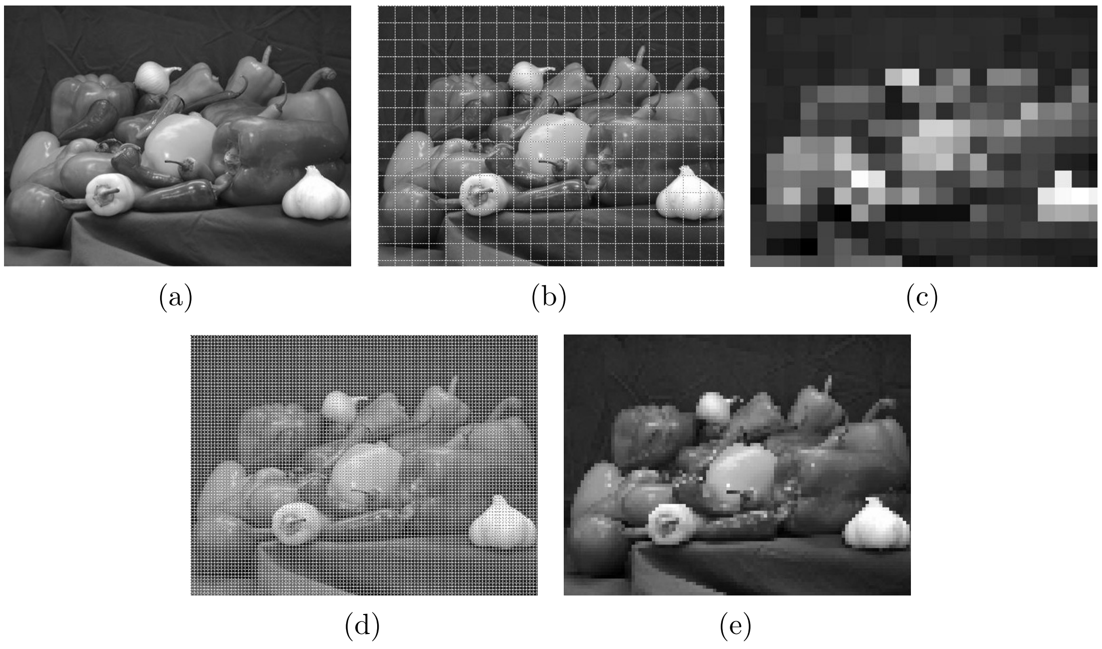
2.1 Digital Image Processing

Digital image processing refers to the process of acquisition, representation and manipulation of digital images. Given an input image, acquired by special input devices, the output of a digitally processed image, is either a report based on image analysis or another image modified by one or more image processing techniques such as, but not limited to, resizing, color transformation, image enhancement, image restoration, compression, object recognition and image segmentation (GONZALEZ; WOODS, 2006).

The image acquisition process is the first main step of a digital image processing pipeline. The acquisition occurs through hardware devices built specifically for this process, including digital cameras and scanners, for example. Once an analog image is acquired, it must be converted and therefore represented as a digital image in a way that is easy to manipulate, process and display by a digital computer (FLORIN; ARSINTE; MASTORAKIS, 2011b).

A digital image is a discrete representation of a visual information that is infinite and continuous in the real world. A digital device is only able to store discrete data, and therefore the continuous image must go through a process called discretization as part of the acquisition process in order to successfully obtain a representation of a scene from the real world in a way that is suitable for digital image processing. Discretization is the process of transforming continuous data into discrete data and is comprised of two steps: sampling and quantization (THYAGARAJAN, 2011). In the context of digital images, sampling determines the spatial resolution of the image while the quantization determines the different color intensity values a image can hold. For example, the different gray levels on grayscale images. An example of different sampling rates is shown in Figure 1. Figure 2 shows the impact of different quantization levels in the digital representation of an image.

Figure 1 – The continuous image (a) sampled through an acquisition system with low-resolution (b) is shown in (c). The same image sampled through an acquisition system with a slightly better resolution (d) is shown in (e).



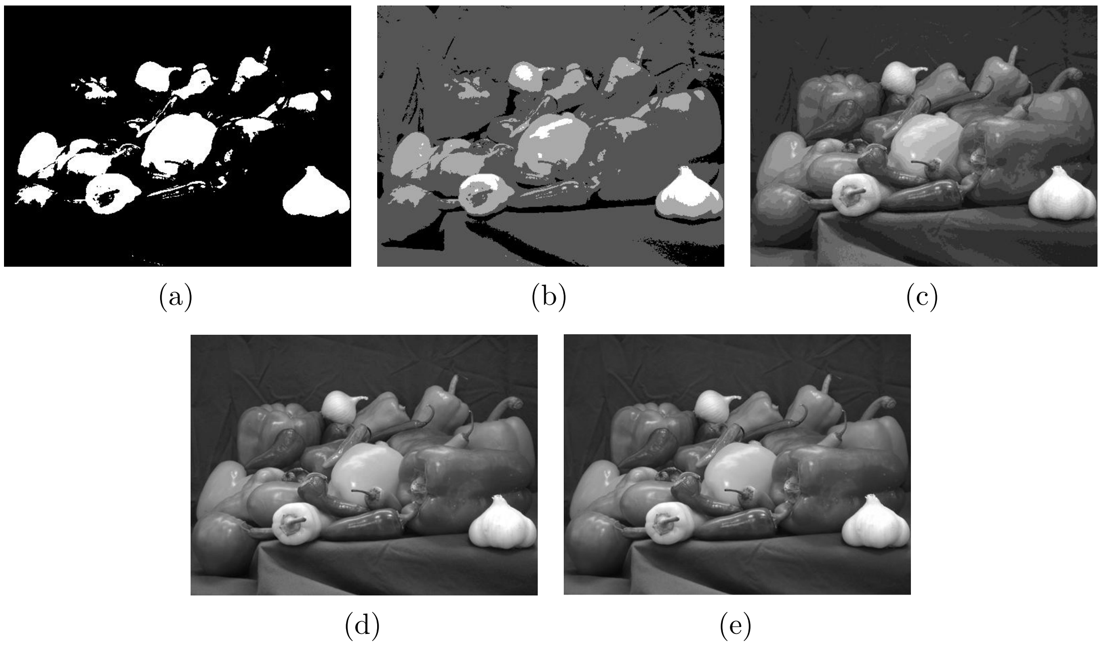
Source: Own authorship.

2.1.1 Representation of digital images

An image can be seen as a two-dimensional function $f(x, y)$ of the light intensity, where x and y are the pixels coordinates of the image. The value of f is proportional to the color intensity of the image in every pixel (LI; DREW; LIU, 2014). The range of the pair of coordinates (x, y) comes from the sampling process, known as image resolution. The range of intensity values each pixel can hold comes from the quantization process. A 4 bits 512x256 image means that x varies between 0 and 511, y between 0 and 255 and each pixel can hold 16 distinct levels of color. Therefore, for any pair of coordinates (x, y) inside this range, $f(x, y)$ gives the color intensity of the pixel in that coordinate. Figure 3 shows a grayscale image overlaid by a 3D plot of its gray levels.

The intensity of each pixel is directly related to the quantization level applied in the acquisition process, a 1 bit image as seen in Figure 2(a), can only hold two distinct values (0 or 1) and therefore results in a black and white image. As we can see in Figure 2 the more bits an image has, better the quality is. Usually, grayscale images are represented with 8 bits and color images with 24 bits. It is important to note that the quality of the image is also directly related to the sampling rate, generally speaking the bigger the resolution is, the better. However, bigger resolution requires more storage space, which is why images are generally compressed in order to reduce its size. In a computer, a grayscale

Figure 2 – (a) Image quantized with 1 bit (two distinct levels), (b) with 2 bits (four distinct levels), (c) with 4 bits (sixteen distinct levels), (d) with 6 bits (64 distinct levels), (e) with 8 bits (256 distinct levels).



Source: Own authorship.

digital image is stored as a two-dimensional matrix of the following form

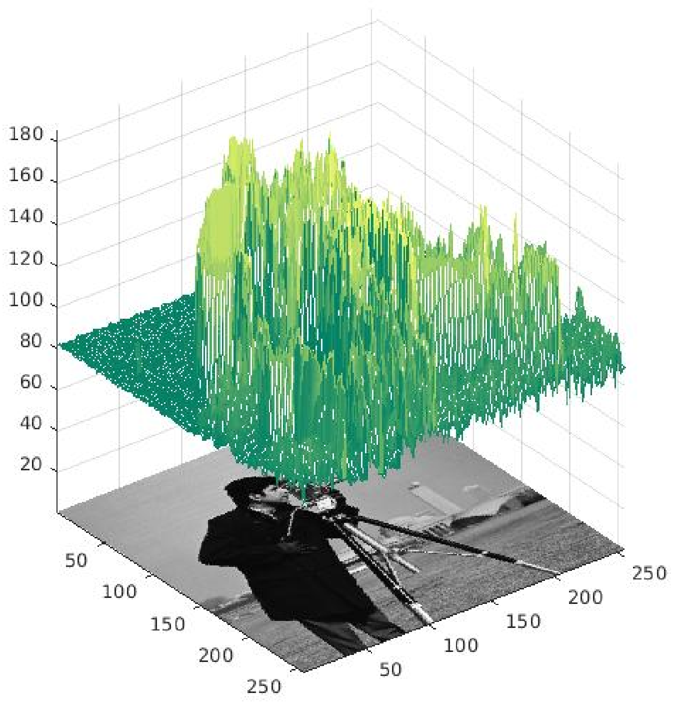
$$\text{f(x,y)} = \underbrace{\begin{bmatrix} 43 & 44 & 46 & ... & 42 & 42 \\ 44 & 44 & 44 & ... & 39 & 39 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 89 & 91 & 91 & ... & 30 & 33 \end{bmatrix}}_{w=512} \left.\vphantom{\begin{bmatrix} 4 \\ 4 \\ \vdots \\ 8 \end{bmatrix}}\right\}{\scriptstyle h=384}$$

where the dimensions of the matrix corresponds to the image resolution. In the above example, the image has dimensions of 512x384 and is a grayscale image with 8 bit per pixel.

2.1.2 Color Images

Color images are an extension of grayscale images and are composed of channels, where the combination of such channels gives the perception of color. There are several color systems that can be used to represent a color image, the most common ones are RGB, YUV, HSL and CMYK (HUNT, 2005). The goal of a color system is to standardize the way colors are specified in a digital manner (WYSZECKI; STILES, 2000).

Figure 3 – A plot of the gray levels of a grayscale image



Source: Own authorship.

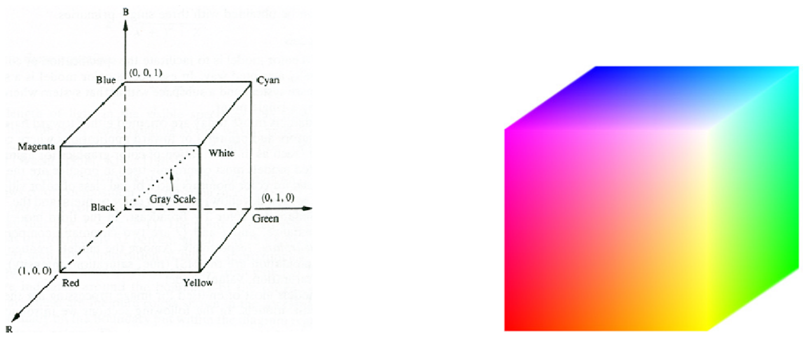
The RGB system is one of the most used color system for digital image processing and represents each pixel in a color image by combining its spectral components red (R), green (G) and blue (B). In this system, each color varies between 0 and 255, which means each color is represented with 8 bits, therefore a total of 16 millions of distinct colors can be represented in this system (LI; DREW; LIU, 2014). Figure 4a shows the so called cube of RGB colors where black is at (0,0,0), white is at (255,255,255). The axis represents the three base colors (red, green and blue) and the other three vertices the primary colors cyano, yellow and magenta. Figure 4b shows a filled cube with the possible color combinations.

Consequently, a color RGB image is formed digitally by the combinations of three color components that are represented by independent matrices (channels) that are combined to give the perception a color image (Figure 5).

2.1.3 Sliding windows

Two important concepts are the pixel neighborhood and the concept of sliding windows. A given pixel p_1 is said to be neighbor of another pixel p_2, if and only if they

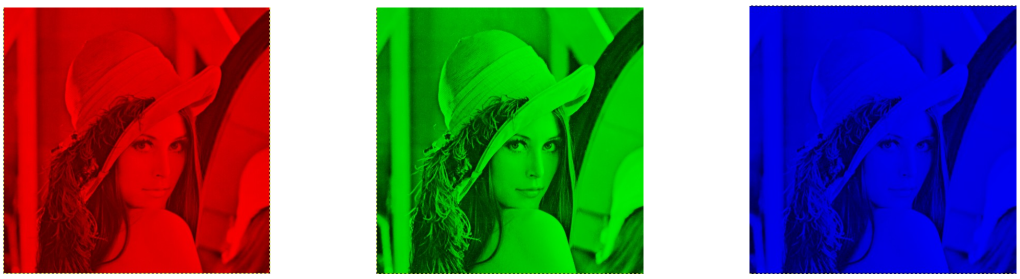
Figure 4 – (a) RGB system. (b) RGB cube of colors.



(a) (b)

Source: Gonzalez e Woods (2006).

Figure 5 – A RGB image decomposed in its channels of color.



(a) R (b) G (c) B

Source: Gonzalez e Woods (2006).

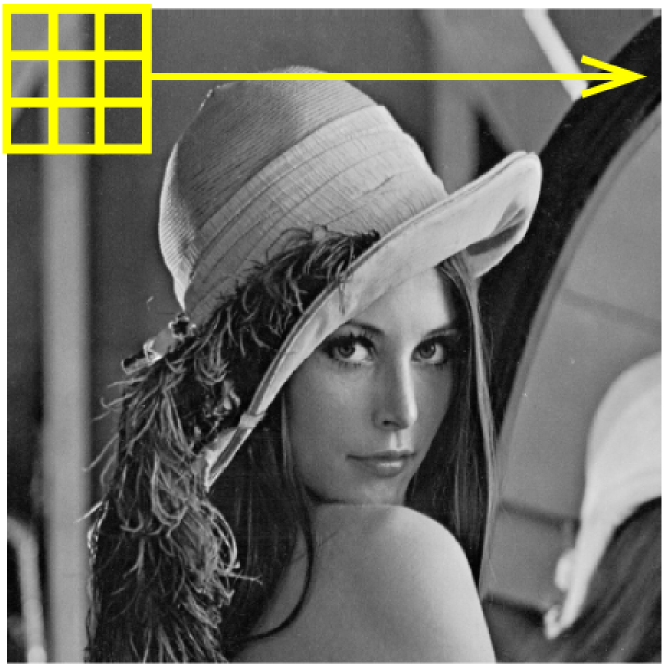
are adjacent horizontally, vertically or diagonally. Thus, a given pixel $p(x, y)$ has a total of eight neighboring pixels, assuming it is not outside of the image borders (Figure 6). An image window refers to a submatrix comprised of the N closest pixels of a reference pixel that is usually at the center of the window. (GONZALEZ; WOODS, 2006). Sliding windows are used for many purposes, from image filtering to object detection. Figure 7 shows how a sliding window works and Figure 8 examples of sliding window sizes. An instance of a sliding window is also usaully called image path.

Figure 6 – The neighborhood of a pixel $p(x, y)$

$p(x-1, y-1)$	$p(x-1, y)$	$p(x-1, y+1)$
$p(x, y-1)$	**p(x,y)**	$p(x, y+1)$
$p(x+1, y-1)$	$p(x+1, y)$	$p(x+1, y+1)$

Source: Own Authorship.

Figure 7 – A sliding window slides through the entire image while some processing happens to the pixel inside the window.



Source: Own Authorship.

Figure 8 – Sliding windows examples.

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & \mathbf{13} & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 \\ 12 & 17 & 34 & 42 & 12 \end{bmatrix}$$

(a) 2x2 sliding window.

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & \mathbf{13} & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 \\ 12 & 17 & 34 & 42 & 12 \end{bmatrix}$$

(b) 3x3 sliding window.

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & \mathbf{13} & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 \\ 12 & 17 & 34 & 42 & 12 \end{bmatrix}$$

(c) 5x5 sliding window.

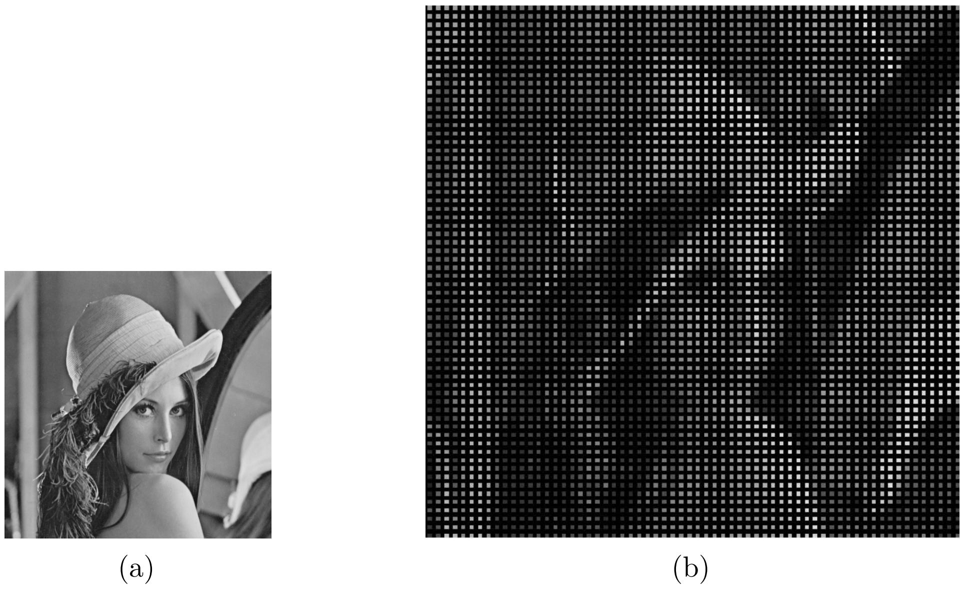
Source: Own Authorship.

In the context of super-resolution algorithms, a sliding window can be used to slide through the image while some process, as part of the SR method, happens inside the sliding windows and estimates the unknown pixels. Usually, interpolation-based methods are performed through a sliding window that traverses the entire image.

2.2 Image Super Resolution

Image Super-Resolution (SR) is an active area of research that aims at generating high-resolution images from one or more low-resolution input images. It is a very important area as it allows to overcome inherent limitations of low-resolution imaging sensors (low-cost phones, surveillance cameras etc). The problem of image SR is that augmenting an image resolution introduces new pixels that must be estimated based solely on the existing pixels of the low-resolution input image(s). Figure 9 shows an image augmented by a factor of 2 with empty pixels, i.e, without any processing to fill in the empty pixels. Therefore, the goal of SR methods is to apply methods to predict the high-resolution image pixels while maintaining the fine characteristics of the image only once.

Figure 9 – (a) Low resolution input image. (b) Output enlarged image without processing.



Source: Own authorship.

Super-Resolution methods can be applied both to two-dimensional images and depth images (LEE; LEE, 2013; SCHUON *et al.*, 2009). However, as we are not dealing with depth images, the remaining of this chapter will focus the discussion on top of two-dimensional images. Within the scope of regular 2D images, SR methods can be categorized in two major classes: Single Image SR and Multi-Image SR. These classes refers merely to the number of images the methods receive as input. Single Image SR methods receives only one low-resolution image as input, whereas Multi-Image SR methods needs multiple low-resolution images as input.

Multi-image SR methods utilize the non-redundant information of multiple image frames of the same scene to generate a high-resolution image. The set of multiple low-resolution images are taken at subpixel misalignment, where each low-resolution image imposes a set of linear constraints on the unknowns high-resolution pixel values. Assuming enough low-resolution input images are provided, it becomes possible to solve the set of linear equations (IRANI; PELEG, 1991; FARSIU *et al.*, 2004). The majority of the Multi-Image SR methods, are limited to a small range of zooming factors (typically factor smaller than 2) due to the complexity of these methods (BAKER; KANADE, 2002; LIN; SHUM, 2004). These limitations lead to the creation of a technique called "image hallucination" or "Example-based Super-Resolution" (BAKER; KANADE, 2000; FREEMAN; PASZTOR, 1999).

Single-Image SR methods, on the other hand, only uses one low-resolution image as input which leads to under-constrained and ill-posed problem. The most common single image SR approaches can be further broken down into two categories: interpolation-based and example-based. Most of the traditional methods found in the literature, such as bilinear and bicubic, fall in the interpolation category, which is based on the assumption that images are linear and smooth. These methods are simple, efficient and to some extend, sufficient to upscale images with decent quality for the average user (GONZALEZ; WOODS, 2006). However, due to discontinuities that exists in natural images, as as the zoom factor increases, the resulting images of these methods are over-smoothed and with visual artifacts that includes blurred edges, ringings and aliasing artifacts (ROSZKOWIAK *et al.*, 2016). There are several other methods in this category including more complex algorithms such as B-splines (UNSER; ALDROUBI; EDEN, 1991) and interpolation based on partial differential equations (KIM; CHA; KIM, 2011; ZHANG *et al.*, 2016).

In example-based approaches, the correspondence between low and high resolution image patches are learned from an external database of low and high resolution images (XIAN; TIAN, 2016). Freeman e Pasztor (1999) was one of the first to propose a learning methodology to compute these relationships. In most cases, example-based methods have been shown to exceed the limitations of classical SR. However, the need for a external database and a lack of relevance between images and a universal training data set has produced results with noise and artifacts along the edges. These methods are also overly complex and computationally expensive, and therefore, have been replaced by single-image methods that can combine the best of both worlds as shown in (GLASNER; BAGON; IRANI, 2009) where a unified framework is proposed for combining classical SR with example-based SR which is able to obtain remarkable high resolution images from as little as a single LR image. The framework is based on the fact that for small image patches in a natural image, self-similarity exists within the image itself and across different resolutions of the same image.

Recently, a third category of SR approach based on machine learning techniques has emerged. Dong *et al.* (2014) proposes a deep learning method for image SR called SRCNN (Super-Resolution Convolutional Neural Network) that directly learns an end-to-end mapping between the pair of low and high resolution images patches. The proposed mapping is built using a deep convolutional network (CNN) that receives the low-resolution images as input and outputs a single high-resolution image. Cui *et al.* (2014) presents a deep learning model called deep network cascade (DNC), which is a cascade of multiple stacked auto-encoders, that gradually upscale low-resolution images layer by layer.

2.2.1 Interpolation-based super-resolution methods

In this section the most common interpolation-based methods are briefly explained. Interpolation methods, remains widely used, despite the artifacts commonly introduced in the interpolated images. They are based on the assumption that images are linear and smooth in terms of its frequency distribution, and the main problem of these methods is that they generally cause a loss on the high frequencies of the images, directly impacting on the sharpness of the images.

These methods often share the same structure, only differing on the interpolation function that is applied to estimate the new pixels. Listing 1 shows how a typical interpolation-based super-resolution method looks like.

Listing 1 A typical algorithm for zooming an image through an interpolation method.

```
Input: N, the scaling factor; I, an input image of size (w,h)
Output: O, zoomed image of size (w*N,h*N)
1: for $y = 1 : h * N$ do
2:     for $x = 1 : w * N$ do
3:         $sx \leftarrow \frac{x}{N}$
4:         $sy \leftarrow \frac{y}{N}$
5:         $O[x, y] \leftarrow f(I, sx, sy)$
6:     end for
7: end for
```

The variables sx and sy, represents a subpixel at coordinate (sx, sy) which is a imaginary pixel that corresponds to the pixel being interpolated in the new, high-resolution image. In the following subsections, the most common strategies for the interpolation function f in line 5 are described in more detail.

2.2.1.1 Nearest replication

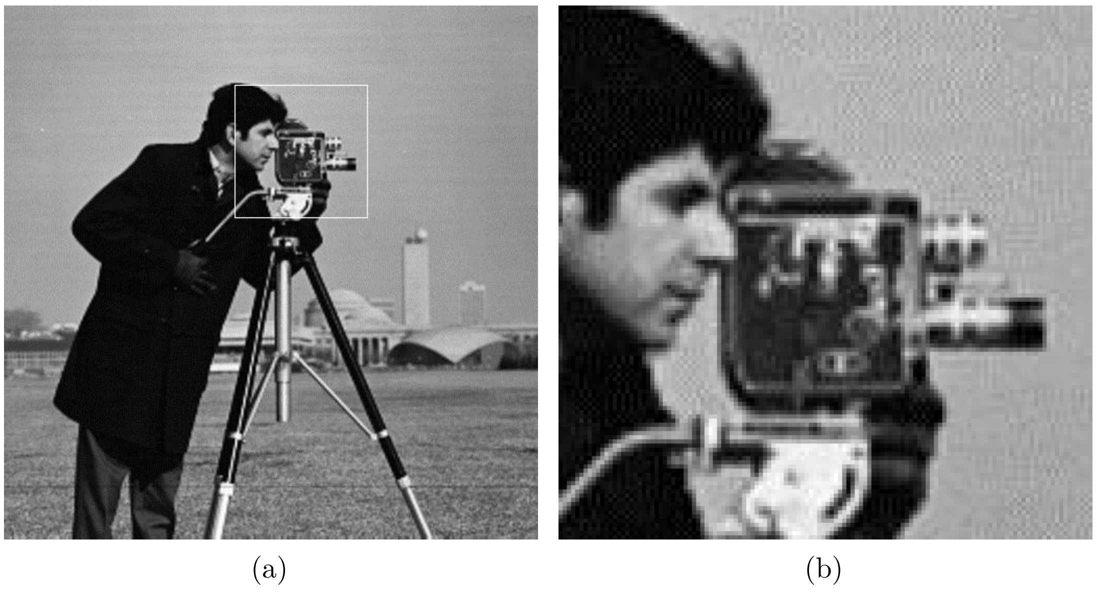
The nearest replication is the most simple method and works solely by replicating the pixels in the expanded neighborhood region. Put in another way, this algorithm simply augments the pixel area by replicating its color intensity across the newly created neighboring pixels. The obvious downside of this method is that zoomed images have a "blocky" appearance due to the pixel area increase. Therefore, the interpolation function simply copies the value of the closest pixel as shown in Equation (2.1)

$$f(I, sx, sy) = I[\lfloor sx \rfloor, \lfloor sy \rfloor], \tag{2.1}$$

where $\lfloor \cdot \rfloor$ rounds $\cdot$ to the nearest integer and I is the original image.

The nearest replication is not vastly used in computer applications as it generally performs worse compared to most of the interpolation methods. The most common problem of this the method is the high introduction of aliasing, specially, around the edges. Figure 10 shows an example of the nearest replication. It is easy to note the aliasing effect on the background and around the borders.

Figure 10 – (a) Original image. (b) The selected slice augmented with nearest replication

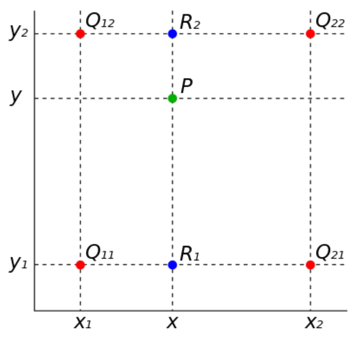

(a) (b)

Source: Own authorship.

2.2.1.2 Bilinear Interpolation

The bilinear interpolation is one of the most popular and used method, due to its speed and simplicity. Not only for zooming images, but also for image rotation (GONZALEZ; WOODS, 2006). This algorithm derives from the standard linear interpolation where any given point in a line can be estimated by a simple rule of three. The bilinear interpolation method extends the standard linear interpolation by interpolation in both x and y direction as shown in Figure 11. According to Pedrini e Schwartz (2008), the point P can be obtained through bilinear interpolation by the Equation given in (2.2). The algorithm uses the closest four pixels (2x2 window) of the current pixel being interpolated.

Figure 11 – On a two-dimensional grid, the point P can be obtained by interpolating in x direction, obtaining R_1 and R_2 and then interpolating them in y direction.



Source: (Bocsika, 2009).

$$f(x', y') = (1 - dx)(1 - dy)f(x, y) + dx(1 - dy)f(x + 1, y) \\ + (1 - dx)dyf(x, y + 1)dxdyf(x + 1, y + 1), \tag{2.2}$$

where (x', y') is the pixel being interpolated and dx, dy the distance between the top-left pixel (x, y) and the one being interpolated, i.e, $dx = x' - x$ and $dy = y' - y$. In the equation, $f(x, y)$ is equivalent to Q_{11} in the figure, $f(x, y + 1)$ is equivalent to Q_{12} and so on.

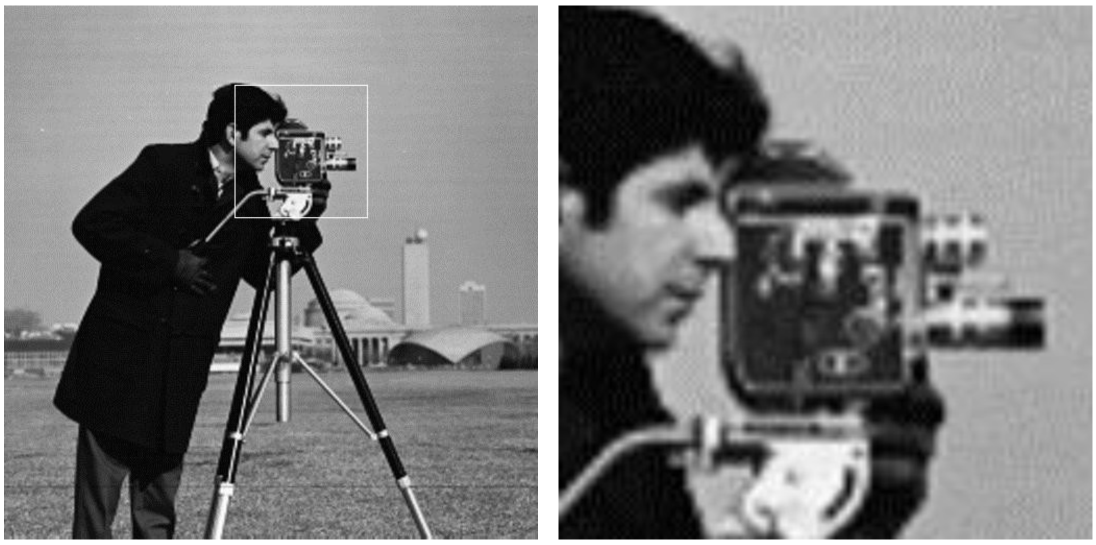
While the bilinear interpolation algorithm presents satisfactory results for the average user, it has one big drawback: as the scaling factor increases, the blurring effect becomes considerably perceptible, reducing the sharpness of the resulting image since the image edges are smoothed. In Figure 12, the augmented image does not have a considerable aliasing effect, when compared to Figure 10. However, it is easy to see that the image became over-smoothed, which tends to reduce the sharpness of the image.

2.2.1.3 Bicubic Interpolation

The bicubic interpolation is an extension of the standard cubic interpolation. Similarly to the bilinear interpolation, bicubic interpolation allows applying the cubic interpolation on a two-dimensional grid. Unlike the bilinear interpolation, this method utilizes the sixteen closest pixels. A weight that represents the distance to the new pixel being interpolated is assigned to each one of these pixels. It often produces better results when compared to bilinear interpolation but still suffer from the same problems, although in a lower magnitude. In this method the interpolated pixel at any given (x', y') position is obtained according to Equation (2.3).

$$f(x', y') = \sum_{i=0}^{3} \sum_{j=0}^{3} a_{ij} x^i y^j, \tag{2.3}$$

Figure 12 – (a) Original image. (b) The selected slice augmented with bilinear interpolation.



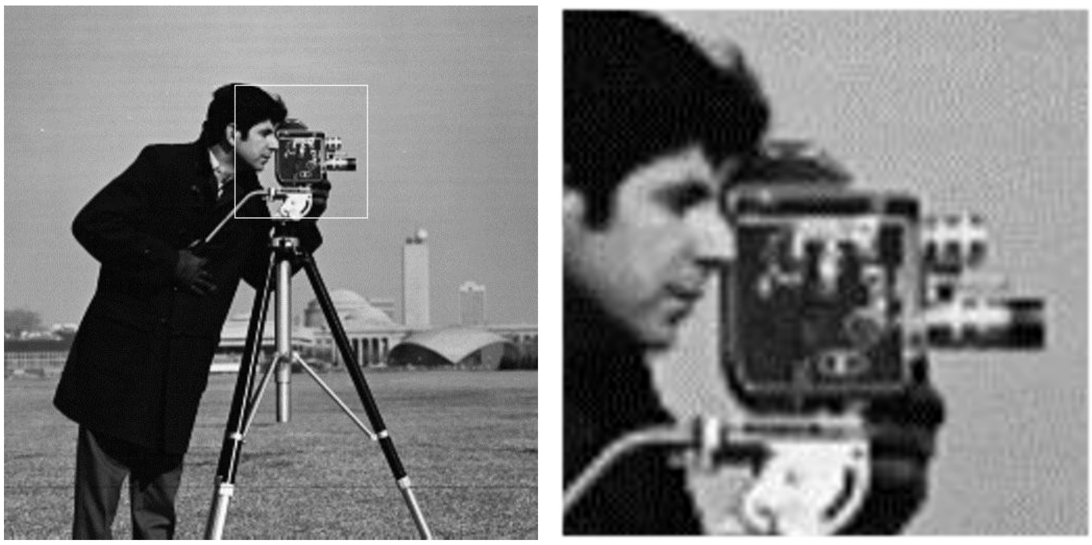
(a) (b)

Source: Own authorship.

where the sixteen coefficients a_{ij} are determined from the same number of equations that can be written from the closest sixteen pixels.

Generally, the bicubic interpolation performs better when compared to bilinear interpolation. It tends to preserve the fine details and sharpness of the image by not over-smoothing the image borders as shown in Figure 13.

Figure 13 – (a) Original image. (b) The selected slice augmented with bicubic interpolation.



(a) (b)

Source: Own authorship.

3 Linear Regression

This chapter presents a brief introduction to linear regression and its mathematic-related topics, including how to run linear regression models and apply transformations to normalize the data prior to the execution of the model. Data transformation is also presented as it is a key part of the proposed super-resolution method, special attention is given to the Box-Cox transformation since it is the transformation used in the proposed method.

3.1 Introduction to Linear Regression

Regression analysis is a technique for investigating and modeling the relationship between variables. There are numerous applications of this technique that span across many different fields, including physics, engineering, economics, data analysis, artificial intelligence and many others. Regression analysis seeks a relationship between response variables and predictor variables (MONTGOMERY; PECK; VINING, 2012). This relationship can be expressed through many kinds of models and once the relationship between the variables has been established, it can then be used to predict the unknown points of interest.

The most common regression model is the linear regression, which at the most basic level, seeks a relationship between one predictor (or regressor) variable x and the response variable y. It is represented by the following equation of a straight line

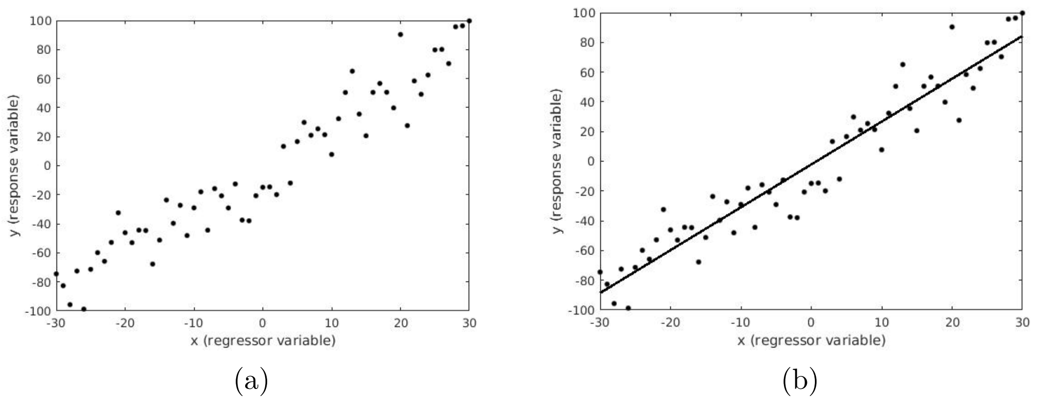
$$\hat{y} = \beta_0 + \beta_1 x + \epsilon, \tag{3.1}$$

where the notation $\hat{y}$ indicates that the value of y is a term. The variable ϵ is considered an random disturbance or error introduced by the model, accounting for the failure of the model to fit the data perfectly. The errors are assumed to have mean zero and unknown variance σ^2. The terms β_0, β_1 are the so called regression coefficients. The regression coefficients can also be seen, respectively, as the intercept and slope of the straight line.

Equation (3.1) is called *simple linear regression* because it only involves one regressor variable (x). The data for the simple linear regression model is often disposed in a scatter diagram such as the one in Figure 14(a). In Figure 14(b) we show the regression line, which is the best fit line for the data distribution, i.e, the straight line that best fits all points, minimizing the error ϵ in Equation (3.1).

In most of the applications of regression, the equations from the regression models are only an approximation of the true relationship between the variables, yet it might be

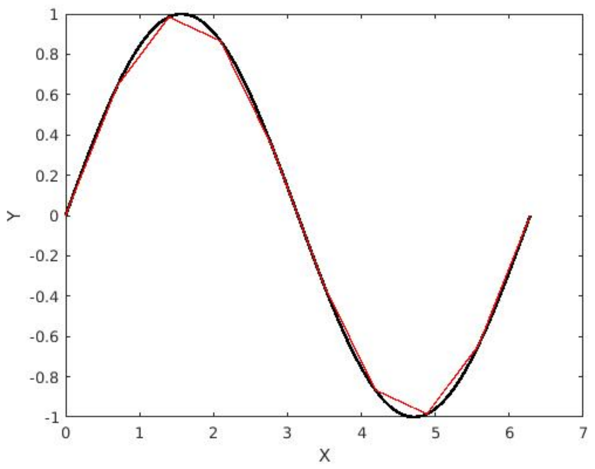
Figure 14 – (a) Scatter diagram for a random set of data. (b) Best fit line between x and y



Source: Own authorship.

acceptable to approximate the true relationship through a linear model. In more complex systems, it is also feasible to make use of a “piecewise linear” regression function to provide a reasonable approximation of such complex relationships (Figure 15).

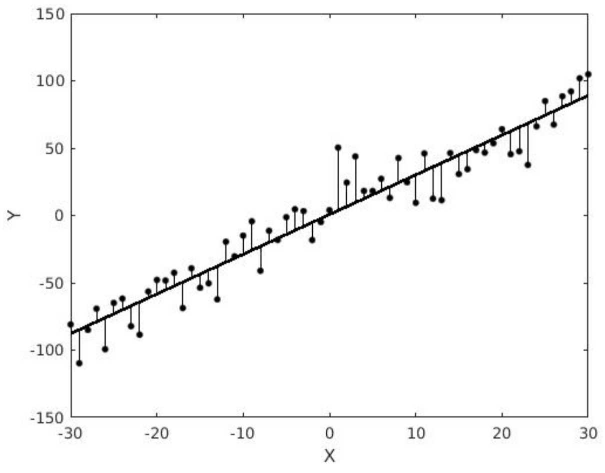
Figure 15 – Piecewise linear approximation of a complex function, using 8 “knots”.



Source: Own authorship.

Obtaining the regression coefficients (the β terms) is one of the most important steps of any linear regression model. The most common method to calculate these values in the simple linear regression model is through the method of least squares (GEER, 2005). The least squares method simply obtain β_0 and β_1 that a straight line and minimize the quadratic sum of the vertical distances from each known point to the straight line. This is equivalent of finding a straight line that best fits all points in the scatter plot. The distance between each point to the straight line is the error of that point if it were estimated using the equation of the line as it is shown in Figure 16 by the vertical distances to the straight line for each known point.

Figure 16 – Vertical distances between the true points and the best fit line.



Source: Own authorship.

These errors are obtained by rewriting (3.1) as

$$\epsilon_i = y_i - \beta_0 - \beta_1 x_i, i = 1, 2..., n. \tag{3.2}$$

The sum of squares of these vertical distances can then be expressed as

$$S(\beta_0, \beta_1) = \sum_{i=1}^{n} (\epsilon_i)^2 = \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i)^2. \tag{3.3}$$

The method of least squares consists of minimizing $S(\beta_0, \beta_1)$, or in other words, consists of obtaining β_0 and β_1 such that the sum of squares of the vertical distances are minimized. The values of β_0 and β_1 that satisfy these constraints are given by

$$\hat{\beta_1} = \frac{\sum_{i=1}^{n} (y_i - \overline{y})(x_i - \overline{x})}{\sum_{i=1}^{n} (x_i - \overline{x})^2}, \tag{3.4}$$

and

$$\hat{\beta_0} = y - \beta_1 \overline{x}, \tag{3.5}$$

where $\overline{x}$ e $\overline{y}$ are, respectively, the means of the variables x_i and y_i, with $i = 1, 2...n$. With the values of β_0 and β_1, which are the least square estimates, we can obtain an approximation of any unknown point of interest by using Equation (3.1).

3.2 Multiple Linear Regression

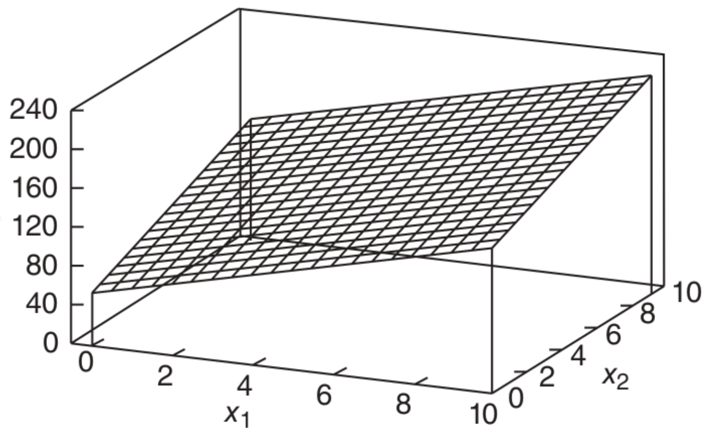
The generalization of the linear regression model is given by Eq (3.6),

$$\hat{y_i} = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + ... + \beta_p x_{pi} + \epsilon, \tag{3.6}$$

where the response variable y is related to p regressors $x_{1i}, x_{2i}, ...x_{pi}$. This generalized model is called *multiple linear regression model* because more than one regressor variable is involved. The model is linear in terms of the parameters $\beta_0, \beta_1, \beta_2, ...\beta_p$ and not because y is a linear function of X (CHATTERJEE; HADI, 2006). This model does not describe a straight line anymore as there are more than two dimensions. For example, the following two-dimensional regression model given by Equation (3.7) defines a plane surface in the three-dimensional space as we can see in Figure 17.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon, \tag{3.7}$$

Figure 17 – An example of a regression plane for the model in Equation (3.7).



Source: Montgomery, Peck e Vining (2012)

For multiple linear regression, it is also possible to express the problem in array notation. Given the following matrices

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, X = \begin{bmatrix} 1 & x_{11} & ... & x_{1p} \\ 1 & x_{21} & ... & x_{2p} \\ \vdots & \vdots & \vdots & \\ 1 & x_{n1} & ... & x_{np} \end{bmatrix}, \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{bmatrix} \epsilon = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix} \tag{3.8}$$

The linear model in (3.1) can be expressed in matrix notation as

$$Y = X\beta + \epsilon, \tag{3.9}$$

where the β coefficients are obtained by minimizing $S(\beta)$ and is given by

$$S(\beta) = (Y - X\beta)^T (Y - X\beta), \tag{3.10}$$

which generates the following system of equations

$$(X^T X)\beta = X^T Y, \tag{3.11}$$

considering that (X^TX) is invertible, $\hat{\beta}$ can be obtained through the follow equation

$$\hat{\beta} = (X^TX)^{-1}X^TY, \tag{3.12}$$

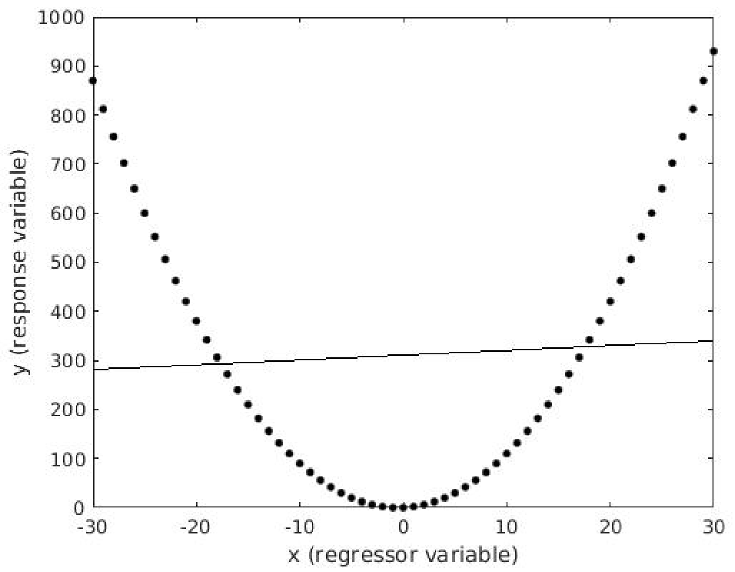
and the prediction of an set of unknown points Y is obtained through

$$\hat{Y} = X\hat{\beta}. \tag{3.13}$$

3.3 Transformations

The assumption that for any set of predictor variables X inside the range of the data, the linear equation (3.6) provides a reasonable approximation of the relationship between y and X is usually the starting point in regression analysis. However as we shall see, this assumption does not hold true for all data distribution, and therefore the need of transformation functions arises. A quadratic function, for instance, will not fit well in a linear model as we can see in Figure 18. While some previous experience or theoretical considerations might indicate to the analyst that the relationship is not linear, there are a few techniques for finding non-linearity such as the lack-of-fit test, scatter diagram, matrix of scatterplots and residual plots (MONTGOMERY; PECK; VINING, 2012).

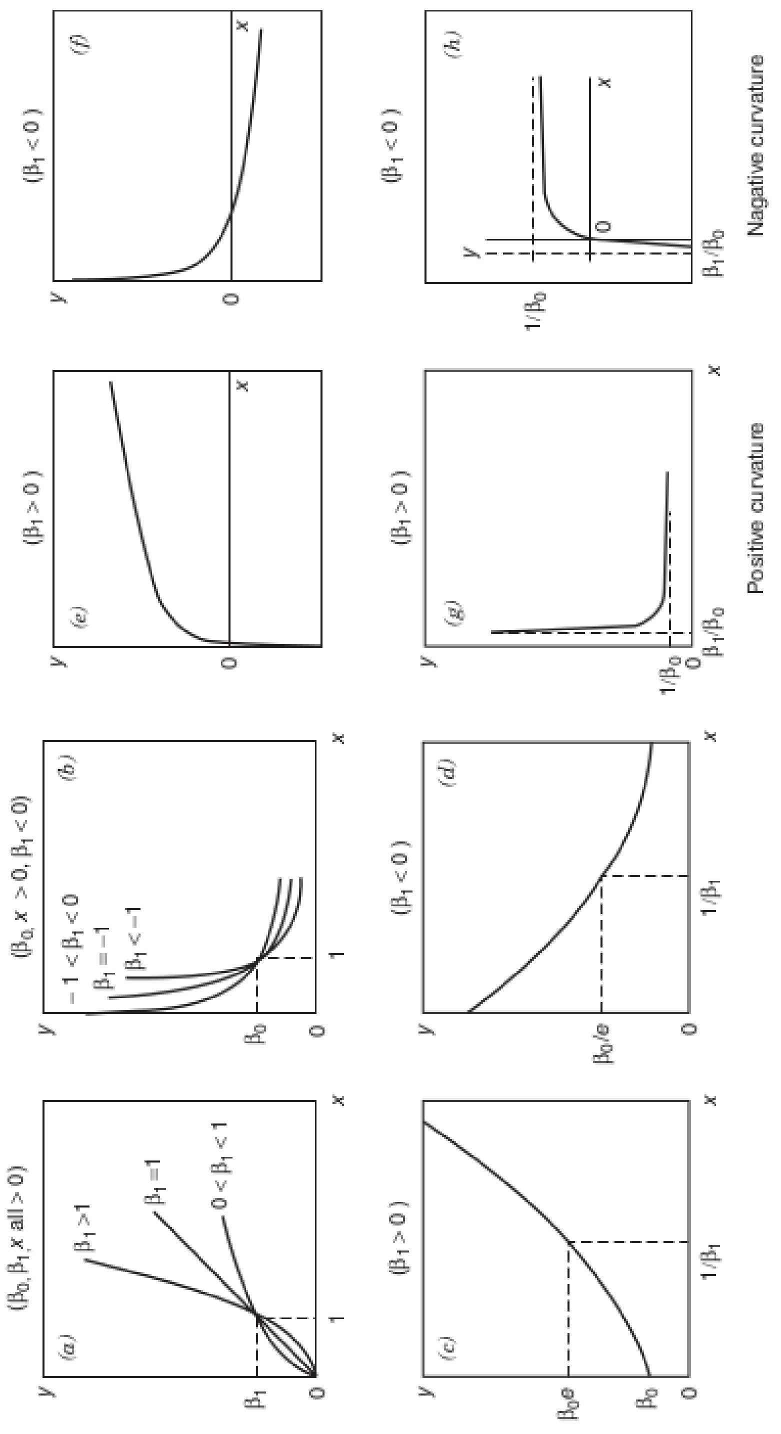
Figure 18 – The best line fit for the function $y = x^2 + x$



Source: Own authorship.

In some cases, non-linear models can be made linear by using linearizable functions. These models are called intrinsically linear and there are several linearizable functions that one might find in the literature. Figure 19 and Table 1 describe a few transformations functions.

Figure 19 – Example of linearizable functions



Source: Daniel e Wood (1980).

Table 1 – Linearizable functions, the corresponding transformations and the resulting linear form.

Figure	Linearizable Function	Transformation	Linear Form
19(a), (b)	$y = \beta_0 x^{\beta_1}$	$T_y = log(y), T_x = log(x)$	$y' = log(\beta_0) + \beta_1 x'$
19(c), (d)	$y = \beta_0 e^{\beta_1 x}$	$T_y = ln(y)$	$y' = ln(\beta_0) + \beta_1 x$
19(e), (f)	$y = \beta_0 + \beta_1 log(x)$	$T_x = log(x)$	$y' = \beta_0 + \beta_1 x'$
19(g), (h)	$y = \frac{x}{\beta_0 x - \beta_1}$	$T_y = \frac{1}{y}, T_x = \frac{1}{x}$	$y' = \beta_0 - \beta_1 x'$

Source: Montgomery, Peck e Vining (2012)

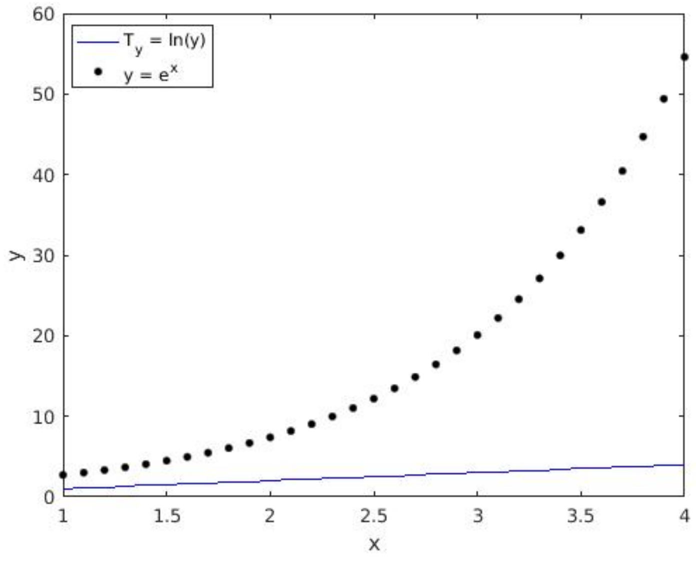
As an example, suppose we want to fit the following nonlinear function through a linear model

$$y = e^x \tag{3.14}$$

It is possible to show that the function is intrinsically linear as it can be transformed to a straight line by the following logarithmic function. The plot for both the function and its transformed counterpart is shown in Figure 20.

$$T_y = ln(y) \tag{3.15}$$

Figure 20 – The function $y = e^x$ linearized by $T_y = ln(y)$



Source: Own authorship.

The linearized version of the function is therefore much more suitable to be fitted by a linear model. It is important to note that the transformation function must be invertible, otherwise it would not be possible to transform back to the original domain of the data.

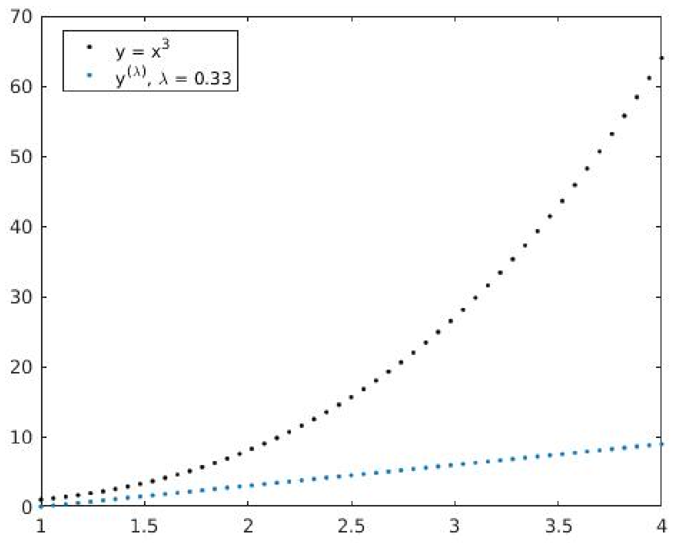
3.3.1 Box-Cox transformation

Box-Cox is a transformation function proposed by Box e Cox (1964). The Box-Cox transformation belongs to the class of power transformations y^λ and consists on the representation of a family of functions, where each family of functions is generated by a λ parameter, which is a parameter to be determined. This transformation is show in Equation (3.16) and its inverse in Equation (3.17). Although not suitable to all applications, the λ parameter can be estimated simultaneously along with the regression parameters using the method of maximum likelihood (BOX; COX, 1964). In chapter 4, we present a method that we developed for finding the best λ parameter that is more suitable to our needs. An example of the application of the Box-Cox transformation is given in Figure 21.

$$y^{(\lambda)} = \begin{cases} \lambda \neq 0, & \frac{y^\lambda - 1}{\lambda} \\ \lambda = 0, & ln(y) \end{cases} \tag{3.16}$$

$$y^{(\lambda)^{-1}} = \begin{cases} \lambda \neq 0, & e^{\frac{log[(\lambda * (y)^{(\lambda)} + 1]}{\lambda}} \\ \lambda = 0, & e^y. \end{cases} \tag{3.17}$$

Figure 21 – Box-Cox applied to the function $y = x^3$ with $\lambda = 0.33$



Source: Own authorship.

In Figure 21, the cubic function $y = x^3$ is plotted alongside with the same function transformed by Box-Cox with $\lambda = 0.33$. With that value for lambda, it is clear to see that the function presents a linear behavior, being more suitable to be fitted by a linear model.

4 Proposed method

In this chapter we describe our novel interpolation-based single image SR algorithm. The major difference between interpolation-based zooming algorithms is the interpolation function, which, given a sliding window size, dictates how the existing image pixels are used to estimate the unknown ones. As briefly mentioned before, interpolation-based methods can be classified as linear and non-linear as well as adaptive and non-adaptive.

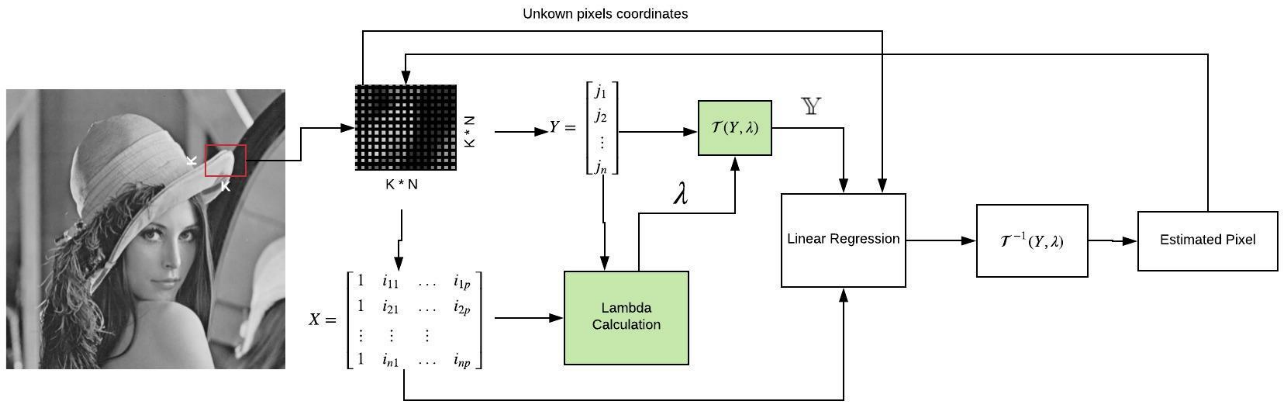
Linear techniques, as the name suggest, treats all pixels in a linear fashion (i.e, as a weighted summation) and therefore are not well suited for most natural images due to the blurring factor applied to all image structures (points, edges, lines etc). Non-linear functions can therefore be introduced to diminish the blurring effect introduced to the resulting image, as the pixels would not be treated as a simple weighted summation, but instead, through a non-linear function. The non-linear function would be chosen based on either the characteristics of the image or based on what it is most important on the image, like the image edges. However, non-linear functions are not a one-size-fits-all solution as a particular non-linear function will not necessarily work well for a wide variety of images. In fact, a non-linear function might not even work well for all regions of the same image (BURGER; BURGE, 2009).

We start off by giving an overview of how the proposed method works and how the concept of linear regression can be used as an interpolation function. After that, we explain the introduction of the Box-Cox transformation in our method with the goal of yielding better looking results and obtain a non-linear and adaptive image zooming algorithm. An interesting fact is that the introduction of the Box-Cox alone is responsible for making our method both non-linear and adaptive. Special attention is also given to the problem of finding a good λ value for the Box-Cox transformation, as it is directly related to the quality of the resulting images.

4.1 Overview

In Figure 22, a flowchart of the proposed algorithm is presented. For simplicity, we assume all images are grayscale, however the method can be easily extended to color images. The proposed algorithm works similarly as traditional interpolation-based methods such as bilinear interpolation. First, a sliding window of size (K, K) is defined, which in our case, we have found that the algorithm works best with $K = 2$, as can be seen in the results chapter. For each offset of the sliding window, X and Y matrices for the linear regression are built. Then, a few special steps (highlighted in green) to find λ and apply Box-Cox are applied (these steps are discussed in depth in the following sections).

Figure 22 – A flowchart of the proposed algorithm. The process indicated above is repeated until all unknown pixels are estimated. K is the size of the sliding window and N is the zooming factor. We have found that the best value for K is 2 as having a bigger sliding window adds contribution from pixels that are relatively too far.


Unkown pixels coordinates
K * N
K * N
$Y = [j_1, j_2, \vdots, j_n]$
$\mathcal{T}(Y,\lambda)$
$\mathbb{Y}$
λ
Linear Regression
$\mathcal{T}^{-1}(Y,\lambda)$
Estimated Pixel
$X = \begin{bmatrix} 1 & i_{11} & \dots & i_{1p} \\ 1 & i_{21} & \dots & i_{2p} \\ \vdots & \vdots & \vdots & \\ 1 & i_{n1} & \dots & i_{np} \end{bmatrix}$
Lambda Calculation

Source: Own authorship.

These special steps are what effectively make our algorithm non-linear and adaptive. It is completely possible to suppress those steps and still obtain a high-resolution version of the image, however the result is considerably worse as we will show in our experimental results.

With the matrices X and T_y, which is the Y matrix with Box-Cox applied we run a standard multiple linear regression model, obtaining the β coefficients and using them to estimate the color intensities for the unknown pixels based on its coordinates. Lastly, the Box-Cox inverse transform is applied to convert it back to the original domain.

4.2 Linear Regression as an interpolation function

A simplified version of the algorithm, without Box-Cox is shown first in order to streamline the understanding of how a linear regression model can be used as an interpolation function. In the next sections the introduction of Box-Cox to yield better-looking results is discussed.

The first step of any regression model is to obtain the β coefficients, and as we are working with two-dimensional data, where each pixel intensity value depends on two variables p_x and p_y, i.e the coordinates of the pixel p, a multiple linear regression model is used. The unknown pixels of interest can therefore be estimated with Equation (4.1)

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_{1i} + \hat{\beta}_2 x_{2i}, \tag{4.1}$$

where x_{1i} and x_{2i} are the (p_x, p_y) coordinates of the inth image pixel. The β coefficients can be easily obtained using the matrix notation (Equation (3.12)) discussed in section 3 and shown in Figure 22. The matrix X holds the independent variables, which in our case are the K pixel coordinates that are in the sliding window and the matrix Y holds the pixel intensity values of the K pixels inside the sliding window.

The simplified version of the proposed algorithm, in the form of an implementation of the interpolation function that was already introduced, is shown in Algorithm 2. The pair (sx, sy) is often referred to as "subpixels", as it is essentially a way to represent an unknown pixel of the augmented image in the original image, however they are not actually real pixels.

Listing 2 Simplified version of the proposed algorithm. Assumes $K = 2$.

```
Input: $(sx, sy)$ the coordinate of the unknown pixel; I, an input image of size (w,h)
Output: The estimated pixel intensity value
1: $dx, dy \leftarrow \lfloor sx \rfloor, \lfloor sy \rfloor$
2: $Y \leftarrow \begin{bmatrix} I[dx, dy] \\ I[dx, dy+1] \\ I[dx+1, dy] \\ I[dx+1, dy+1] \end{bmatrix}$
3: $H = \frac{1}{4} \begin{bmatrix} 3 & 1 & 1 & -1 \\ -2 & -2 & 2 & 2 \\ -2 & 2 & -2 & 2 \end{bmatrix}$
4: $\hat{\beta} \leftarrow HY$
5: return $\hat{\beta}_0 + \hat{\beta}_1(sx - dx) + \hat{\beta}_2(sy - dy)$
```

The interpolation function receives the sub-pixel coordinates, which directly relates to an unknown pixel in the augmented image. In order to obtain the sliding window of size $(K; K)$ we simply get the nearest four pixels, assuming the sub-pixel of coordinates $(sx; sy)$ is at the top left coordinate of the sliding window. The variable H is defined as follows

$$H = (X^T X)^{-1} X^T \tag{4.2}$$

where X is defined based on the nearest four pixels of the pixel being estimated.

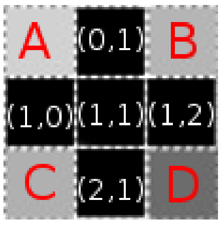
To optimize the algorithm, the X matrix of the first sliding window is fixed and used in all iterations. The resulting pixel is then, translated to its real coordinate. The matrix X for the first sliding window is depicted in Equation (4.3).

$$X = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \tag{4.3}$$

The H matrix shown in line 3 of Algorithm 2 is obtained after calculating Equation (4.2) with the above X matrix. This optimization allows us to avoid several matrix multiplications and thus speeding up the execution of the method. To translate the interpolated pixel to its real coordinate, the real pixel coordinate is defined as $(sx - dx; sy - dy)$.

Figure 23 represents a slice of an augmented image by a factor of 2 with pixels yet to be estimated, the pixel with coordinates $(1; 1)$ would be mapped to the sub-pixel of coordinates $(0,5; 0,5)$ in the original image and the pixels A, B, C and D would be used in the regression model.

Figure 23 – A slice of an augmented image by a factor of 2 with pixels yet to be estimated. Unknown pixels are represented in black.



Source: Own authorship.

4.3 Introducing Box-Cox

There is no guarantee that pixel intensity values for natural images varies linearly, thus having a high Likelihood of introducing a considerable error ϵ in the standard regression model and hence, producing a noisy augmented image. A potential solution to this issue would be to use a n-degree polynomial to better fit the pixel intensity plane, however finding such polynomial introduces a much more complex problem. Our approach to this issue is to use a transformation function to linearize the data prior the regression itself. The transformation function must be invertible, otherwise it would not be possible to reconstruct the image in its original domain.

Box-Cox was chosen due to its simplicity and for being easily invertible without introducing a significant numeric error. However, Box-Cox depends on a parameter called lambda (λ) which the near-optimal value varies based on the data. Traditionally, the Maximum Likelihood and Bayesian methods have been used to estimate the λ parameter.

The Algorithm 3 presents the final version of BCZ. In line 4, we first obtain a good obtain a good estimate for λ. Explanation of the actual method employed to obtain an appropriate λ parameter is described in the next section. Line 5 and 8 is where Box-Cox comes into play, transforming the Y matrix (i.e the pixel intensity values) with Box-Cox and converting it back to the original domain. Note that the estimated pixel value (line 7) is in the Box-Cox domain.

Listing 3 The BCZ method. Assumes $K = 2$.

```
Input: $(sx, sy)$ the coordinate of the unknown pixel; I, an input image of size (w,h)
Output: The estimated pixel intensity value
1: $dx, dy \leftarrow \lfloor sx \rfloor, \lfloor sy \rfloor$
2: $Y \leftarrow \begin{bmatrix} I[dx, dy] \\ I[dx, dy+1] \\ I[dx+1, dy] \\ I[dx+1, dy+1] \end{bmatrix}$
3: $H = \frac{1}{4} \begin{bmatrix} 3 & 1 & 1 & -1 \\ -2 & -2 & 2 & 2 \\ -2 & 2 & -2 & 2 \end{bmatrix}$
4: $\lambda \leftarrow GetLambda(X_0, Y)$
5: $\mathbb{Y} \leftarrow \mathcal{T}(Y, \lambda)$
6: $\hat{\beta} \leftarrow H\mathbb{Y}$
7: $T_p \leftarrow \hat{\beta}_0 + \hat{\beta}_1(sx - dx) + \hat{\beta}_2(sy - dy)$
8: return $\mathcal{T}^{-1}(T_p, \lambda)$
```

In line 3, the best lambda is obtained based on the first sliding window (Equation (4.3)), which is being denoted as X_0. $\mathcal{T}$ is the Box-Cox transformation function.

4.4 Finding a good value for λ

Our goal is to find a lambda that introduces the smallest residual error in the regression. A naive approach for finding such lambda is to do an extensive search across a specific interval of λ values and look for the λ that when used to estimate Y (the response variable) generates the smallest error. This naive approach, as we shall see, poses some challenges and due to that we developed a separate method for finding the best lambda.

The error from a random λ is given by Equation (4.4)

$$\varepsilon = \mathbb{Y} - \hat{Y}, \tag{4.4}$$

where $\hat{Y}$ is an estimate of Y made by the linear regression model with Box-Cox applied using λ as parameter and $\mathbb{Y} = \mathcal{T}(Y, \lambda)$ Let us redefine the variable H as show in Equation (4.5)

$$H = X(X^T X)^{-1} X^T, \tag{4.5}$$

$\hat{Y}$ can then, be expressed in terms of H and $\mathbb{Y}$ according to Equation (4.6).

$$\hat{Y} = H\mathbb{Y} \tag{4.6}$$

which leads to

$$\varepsilon = \mathbb{Y} - H\mathbb{Y}, \tag{4.7}$$

where ε is an column array with the error from every y_i in the response variable. For convenience we rewrite Equation (4.7) as follows

$$\begin{aligned}\varepsilon &= I\mathbb{Y} - H\mathbb{Y} \\ &= (I - H)\mathbb{Y},\end{aligned} \tag{4.8}$$

where I is the identity matrix and $I\mathbb{Y} = \mathbb{Y}$ by the multiplicative identity property (BERNSTEIN, 2005).

Taking the squared error as in Equation (4.9), we can obtain a scalar number representing the error introduced by the lambda parameter.

$$\epsilon = \varepsilon^T \varepsilon \tag{4.9}$$

Algorithm 4 presents a naive approach for finding a more suitable lambda parameter by performing an extensive search across a pre-defined range of lambda values.

Listing 4 A naive approach for guessing the best λ.

Input: X: The regressor variables in array notation. Y: The response variable in array notation. (λ_s, λ_e): The range to perform the search in. p: The precision of the search.
Output: Λ: The best lambda found

```
1:  $min\_err \leftarrow 1 \times 10^{-4}$
2:  $H \leftarrow X(X^T X)^{-1} X^T$
3:  for $\lambda = \lambda_s; \lambda \leq \lambda_e; \lambda = \lambda + p$ do
4:      $\varepsilon \leftarrow (I - H)\mathbb{Y}$
5:      $err \leftarrow \varepsilon^T \varepsilon$
6:      if $min\_err > err$ then
7:          $\Lambda \leftarrow \lambda$
8:          $min\_err \leftarrow err$
9:      end if
10: end for
11: return $\Lambda$
```

The naive approach also introduces the challenge of specifying a reasonable range (λ_s, λ_e) to perform the search in. There is also the expensive computational cost to run this method for every pixel in the high-resolution image. It also does not guarantee a near-optimal value for lambda as the range the search happens in is arbitrarily defined. Therefore, we need a mathematical formula that allows us to estimate a lambda value that introduces the least residual error.

We obtain such formula, through the LSE method, by minimizing (4.9), which contains the square of residual errors, and solving for λ. Expanding ε with Equation (4.7) we obtain

$$S = [(I - H)\mathbb{Y}]^T [(I - H)\mathbb{Y}] \tag{4.10}$$

which can be reorganized as in Equation (4.11) due to the property that states that $(AB)^T = (B^T A^T)$ (BERNSTEIN, 2005),

$$S = \mathbb{Y}^T (I - H)^T (I - H)\mathbb{Y} \tag{4.11}$$

Since $(I - H)$ is both a symmetric and idempotent matrix (KUPPERMAN, 1975; RAWLINGS; PANTULA; DICKEY, 2001) we can further simplify this equation to the following

$$\begin{aligned} S =& \mathbb{Y}^T (I - H)(I - H)\mathbb{Y} \\ =& \mathbb{Y}^T (I - H)\mathbb{Y} \end{aligned} \tag{4.12}$$

Expanding the transformation from Equation (3.16) and assuming Y is strictly greater than zero, we obtain

$$\begin{aligned} S =& \left[\frac{Y^\lambda - \mathbb{1}}{\lambda}\right]^T (I - H) \left[\frac{Y^\lambda - \mathbb{1}}{\lambda}\right] \\ S\lambda^2 =& \left[Y^\lambda - \mathbb{1}\right]^T (I - H) \left[Y^\lambda - \mathbb{1}\right], \end{aligned} \tag{4.13}$$

where $\mathbb{1}$ is a column vector of the same size as Y.

4.4.1 Solving the equation with the Newton-Raphson method

In order to solve Equation (4.13) for λ the Newton-Raphson method is used (NELSON, 1954). Newton's method is a powerful interactive root-finding technique for numerically solving equations. Due to its efficiency and simplicity it is widely used. Let $f(x)$ be the function we desire to find the roots for and r be the root of $f(x) = 0$. The Newton-Raphson method starts by guessing an estimate of r called x_0. From x_0 a new iteration is made, producing a new and hopefully better estimate x_1. From x_1, a new estimate x_2 is produced and so on. The interactions continue until x_i is close enough to r (BEN-ISRAEL, 1966) or no improvement is made.

Considering that x_0 is a good initial guess of the true root r and that $r = x_0 + h$, h measures how far the initial estimate x_0 is from the true root r. The tangent line can be use to obtain the approximation in Equation (4.14), assuming h is sufficiently small.

$$f(r) = f(x_0 + h) \approx f(x_0) + hf'(x_0), \tag{4.14}$$

which leads to Equation (4.15).

$$h \approx -\frac{f(x_0)}{f'(x_0)} \tag{4.15}$$

The approximation of the true root r can then be expressed at each interaction as in Equation (4.16). The interactive process can be executed for a fixed number of times or when $x_{(i+1)}$ and x_i does not change significantly.

$$x_{(i+1)} = x_i - \frac{f(x_i)}{f'(x_i)} \tag{4.16}$$

An algorithm for the Newton-Raphson method for obtaining the best lambda is shown in Listing 5.

Listing 5 Newton-Raphson method for finding the best lambda given X and Y.

```
Input: S(x) and S'(x)
Output: $\Lambda$: The best lambda found
1: $\lambda \leftarrow 1$
2: for i=1:10 do
3:     $\Lambda \leftarrow \lambda - \frac{S(\lambda)}{S'(\lambda)}$
4:     if $|\lambda - \Lambda| < threshold$ then
5:         break
6:     end if
7:     $\lambda \leftarrow \Lambda$
8: end for
9: return $\Lambda$
```

For solving (4.13) using the Newton-Raphson method as in Listing 5, we must derive S with respect to λ (4.17), to obtain S' (4.18) using matrix differential calculus (MAGNUS; NEUDECKER, 1999).

$$\begin{aligned}\frac{\partial S}{\partial \lambda} =& \Big[Y^{\lambda} \odot ln(Y)\Big]^{T}(I-H)\Big[Y^{\lambda} - \mathbb{1}\Big] + \\ & \Big[Y^{\lambda} - \mathbb{1}\Big]^{T}(I-H)\Big[Y^{\lambda} \odot ln(Y)\Big],\end{aligned} \tag{4.17}$$

$$S' = \frac{2\Big[Y^{\lambda} \odot ln(Y)\Big]^{T}(I-H)\Big[Y^{\lambda} - \mathbb{1}\Big] - 2\lambda S}{\lambda^2}, \tag{4.18}$$

where $\odot$ is the hadamard product of matrices (STYAN, 1973). The algorithm in Listing 5 receives the functions (4.17) and (4.18) as input and return the best lambda found. The initial guess x_0 is set to one and the number of interactions is limited to ten. If the difference between x_{i+1} and x_i is less than a sufficiently small threshold, then algorithms bails out early.

5 Experimental Evaluation

We conducted several experimental tests with our proposed method to compare it to the following methods: Neighborhood Interpolation, Bilinear Interpolation and Bicubic Interpolation. These methods, described in detail in section 2.2.1, are often used as benchmarks when evaluating new super-resolution methods. The tests were conducted using 4K images (3840x2160). Two classic evaluation metrics, PSNR and SSIM, used vastly in the literature, drive the comparison and evaluation of our method. To provide a reliable statistical comparison confidence intervals are obtained using the bootstrap method to statistically compare the proposed method against the selected methods.

5.1 Evaluation Metrics

There are two types of image evaluation methods, subjective and objective methods (LAHOULOU *et al.*, 2011). Subjective methods are based solely on human judgment, while objective methods on the other hands are based on a certain numerical criteria to assess the quality of images (HORE; ZIOU, 2010). To establish a numeric method for comparing and measuring the performance of the proposed method, it is necessary to make use of objective image evaluation metrics. For the experimental tests, two image quality metric are going to be used, the PSNR and SSIM metrics.

Overall, image quality assessment methods are not bulletproofs methods. The mathematical models used by most of them are not sophisticated enough to accurately describe the Human Visual System (HVS). The proper understanding of the HVS and how it translates into how humans perceive the image quality, is crucial for a good image quality metric. Unfortunately, most of the existing metrics for evaluating image quality is still limited, making the comparison of super-resolution methods difficult.

5.1.1 PSNR

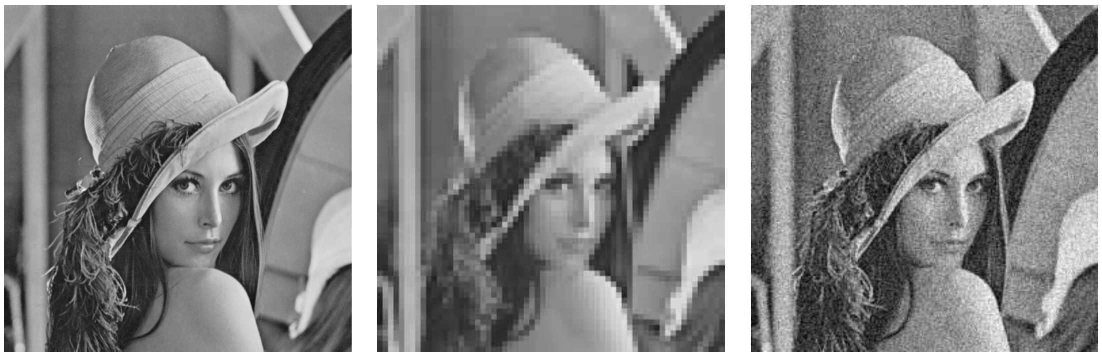
The simplest and most widely used full-reference image quality metric is the mean squared error (MSE), which is computed by averaging the squared intensity differences of degraded and reference image pixels, along with the related quantity of peak signal-to-noise ratio (PSNR) (HORE; ZIOU, 2010). The MSE is defined in Equation (5.1).

$$MSE = \frac{1}{MN}\sum_{n=1}^{M}\sum_{m=1}^{N}[f(x,y) - g(x,y)]^2, \tag{5.1}$$

Both the MSE and PSNR are easy to calculate and have clear physical meaning, however they do not appropriately match the perceived visual quality (GIROD, 1993; TEO; HEEGER, 1994; WANG; BOVIK; LU, 2002).

Dourado (2014) provides a good example of such problem in Figure 24, where it demonstrates a situation in which the MSE metric failed to match the perceived visual quality. Figure 24(b) is a image reconstructed with the nearest interpolation, while Figure 24(c) is the original image with the addition of gaussian noise. In spite of that, the two distorted image have the same value for the MSE.

Figure 24 – The lena image with two distinct distortions.



(a) Original image (b) Nearest interpolation (c) Gaussian noise

Source: Dourado (2014).

The PSNR and MSR are referred to as "full-reference" metrics because a complete reference image is assumed to be known and available as the ground truth image. Given an image $f(x, y)$ and a distorted image $g(x, y)$ of same size, the PSNR is defined in terms of the MSE and its formula is shown in Equations (5.2).

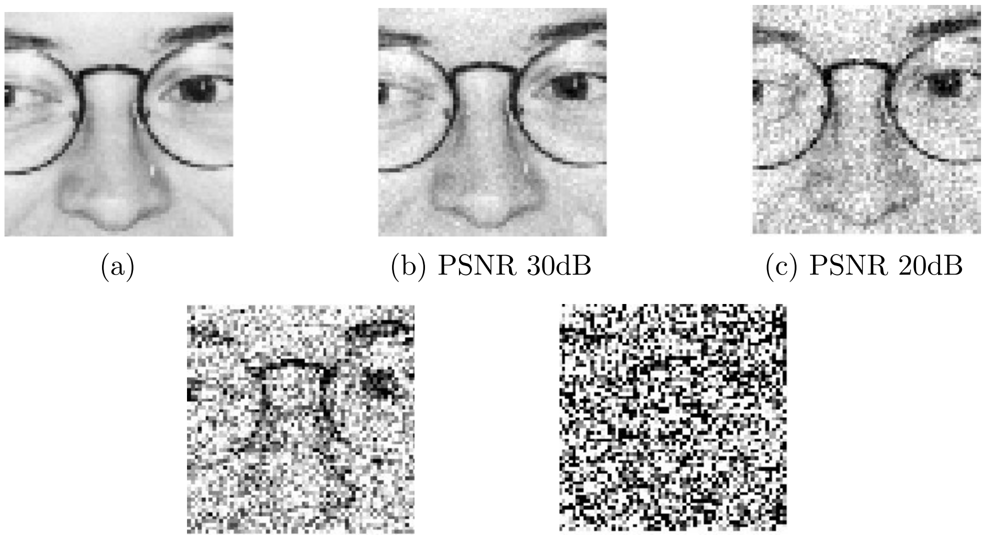
$$PSNR = -10log_{10}\frac{MSE}{S^2}, \tag{5.2}$$

where S is the highest pixel intensity value. The PSNR is measured in decibels (dB). The PSNR tends to infinity as MSE approaches to zero. This means that the bigger the value, the better the quality of the reconstructed image is. Figure 25 shows a comparison of several PSNR measurements from the same image with various level of noise.

5.1.2 SSIM

The structural similarity index (SSIM) is well-known and vastly used full-reference image quality assessment framework and it measures the structural similarity between two images (WANG *et al.*, 2004). It is an attempt to better match the perceived visual quality of the HVS. This framework assumes that natural images are highly structured, i.e, the pixels are highly dependent and carry important information about the objects in the image, especially when they are close to each other.

Figure 25 – (a) The reference image. (b), (c), (d) and (e) are versions of the same image with different level of distortion.



(a) (b) PSNR 30dB (c) PSNR 20dB

(d) PSNR 10dB (e) PSNR 0dB

Source: Todd Veldhuizen (1998).

The SSIM tries to express the HVS in a mathematical model and instead of assuming that the error is directly expressed homogeneously in the image, it provides a model for the image distortion, from the perspective of image formation. This image distortion model is build as the combination of three components, the loss of correlation, luminance distortion and contrast distortion. The diagram for this model is shown in Figure 26.

Given a reference image $f(x, y)$ and a distorted image $g(x, y)$ of same size, the SSIM is defined as a combination of the three components as seen in Equation (5.3)

$$SSIM(f, g) = l(f, g)^{\alpha} c(x, y)^{\beta} s(x, y)^{\gamma}, \tag{5.3}$$

where $\alpha > 0, \beta > 0$ and $\gamma > 0$ are parameters used to adjust the relative importance of each of the three components.

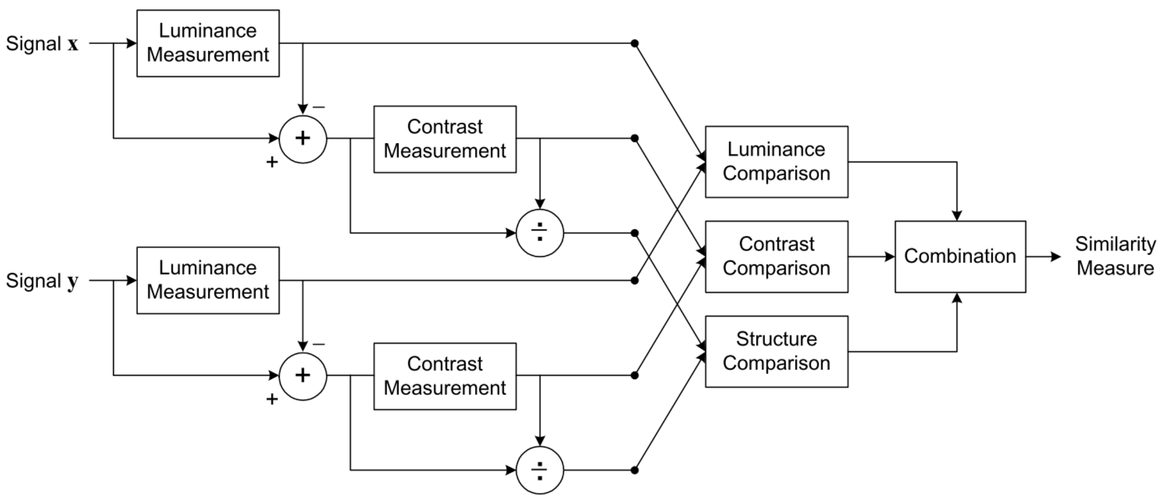
The $l(f, g)$ component is the luminance comparison function, it is a function of μ_f and μ_g and is defined as follows

$$l(x, y) = \frac{2\mu_f \mu_g + C_1}{\mu_f^2 + \mu_g^2 + C_1}, \tag{5.4}$$

where μ_x is estimated as the mean intensity and is defined in Equation (5.5). C_1 is a positive constant to prevent numerical instabilities when $\mu_f^2 + \mu_g^2$ is close to zero.

$$\mu_x = \frac{1}{N} \sum_{i=1}^{N} x_i \tag{5.5}$$

Figure 26 – Diagram of the SSIM method



Source: Wang *et al.* (2004)

The $c(f, g)$ component is the contrast comparison function. It is a function of σ_f and σ_g and is defined in Equation (5.6)

$$c(f,g) = \frac{2\sigma_f\sigma_g + C_2}{\sigma_f^2 + \sigma_g^2 + C_2}, \tag{5.6}$$

where σ_x is the standard deviation which can be expressed as shown in Equation (5.7) (JAMES *et al.*, 2014).

$$\sigma_x = \left(\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)^2\right)^{\frac{1}{2}} \tag{5.7}$$

The last component $s(f, g)$ is the structure comparison function and is defined in Equation (5.8)

$$s(f,g) = \frac{\sigma_{fg} + C_3}{\sigma_f\sigma_g + C_3}, \tag{5.8}$$

where σ_{xy} is obtained through Equation (5.9)

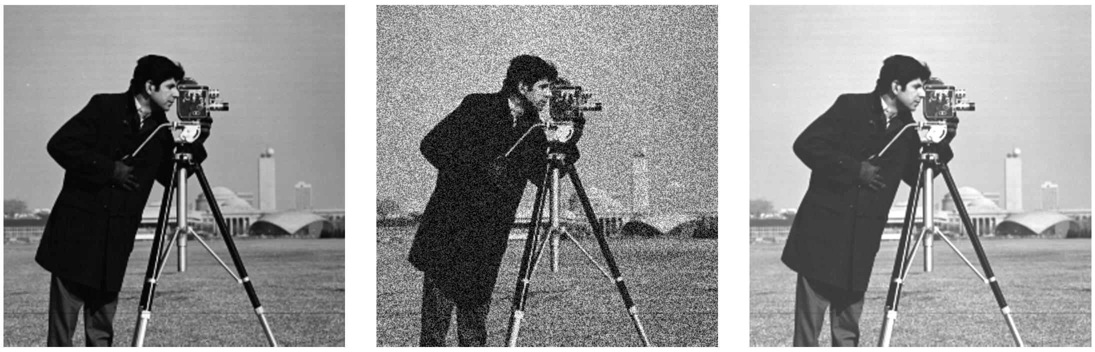
$$\sigma_{xy} = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)(y_i - \mu_y) \tag{5.9}$$

Usually, the constants are set as $\alpha = \beta = \gamma > 1$ and $C_3 = \frac{C_2}{2}$, which leads to the standard form of the SSIM show in Equation (5.10).

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{5.10}$$

The value returned by the SSIM is a decimal number between -1 and 1, where -1 is indicates that the two images are completely different. On the other hand if the SSIM index for two images is equal to 1, the images are identical. Figure 27 shows the SSIM value for distinct distortions applied to a reference image.

Figure 27 – (a) The reference image. In (b) the image with random noise. In (c) the image added with a constant number.



(a) (b) SSIM = 0.14 (c) SSIM = 0.84

Source: Wang e Bovik (2009).

According to extensive research carried by Dourado (2014), the SSIM has proven to outperform other traditional metrics when it comes to match the true perceived visual quality. For better results, Wang *et al.* (2004) suggests that the test and reference image are broken down into several blocks and the framework must be applied to each block, instead of a one full pass at the image.

5.2 Bootstrap validation

Bootstrap is a fast and flexible computed-based statistic technique for estimating quantities about a population by averaging estimates from multiple small data samples (MARTINEZ; MARTINEZ, 2001; EFRON; TIBSHIRANI, 1986). The bootstrap method does not make any assumptions about the underlying population and uses the random samples as an estimate of the population. The sampled population is called the empirical distribution, which we will be denoting as Φ. Each x_i has the same probability $\frac{1}{n}$ of being selected when a new sample is generated from Φ, i.e, each x_i has the same likelihood of being selected. It does not mean however, that all elements of the sample must or will be selected.

Given a statistic $s(\cdot)$, the bootstrap method utilizes probability theory to allow the analyst to obtain inferences of the sample using $s(\cdot)$ and the re-sampled sets. The bootstrap method can be summarized as follows: (EFRON; TIBSHIRANI, 1994)

1. Given a random sample of a population $\Phi = \{x_1, x_2, ..., x_n\}$.
2. Resample Φ with replacement to obtain $\Phi_b^* = \{x_1^{*b}, x_2^{*b}, ..., x_n^{*b}\}$ of size n.
3. Calculate the same statistics $s(\cdot)$ using the bootstrap sample to obtain, $\hat{\theta}_b^*$.
4. Repeat 2 and 3 B times.
5. Use the bootstrap replicas $(\hat{\theta}_b^*)$ to get the desired statistical characteristics.

The notation Φ_b^*, with $b = 1, ..., B$, is the b-th bootstrap data set, and is assumed to be a good representation of the original sample Φ. $\hat{\theta}_b^*$, with $b = 1, ..., B$, is the bootstrap replica of the b-th sample, obtained through $s(\Phi_b^*)$. B is a parameter of the method, fixed by the analyst and determines how many bootstrap replicas will be generated.

A confidence interval represents the range of values a given parameter can take, considering a fixed error. The confidence interval can be used as a tool to compare distinct methods of interest and we use them as a way to interpret how, on average, our method performs against the selected methods. Consider two confidence intervals c_1 and c_2 obtained for the PSNR mean p_1 and p_2 of two distinct SR methods SR_1 and SR_2 and the following test of hypothesis

$$\begin{cases} H_0 : p_1 = p_2 \\ H_1 : p_1 \neq p_2 \end{cases} \tag{5.11}$$

H_0 is not rejected, if and only if, the confidence intervals c_1 and c_2 overlaps. If H_0 is not reject it is assumed that the methods SR_1 and SR_2 have the same performance, on average, according to the PSNR metric. On the other hand, if c_1 and c_2 do not overlap, H_0 is not accepted, meaning that SR_1 and SR_2 have distinct performance. The following test of hypothesis can also be stated

$$\begin{cases} H_0 : p_1 \geq p_2 \\ H_1 : p_1 < p_2 \end{cases} \tag{5.12}$$

The hypothesis H_0 is not rejected if the interval c_1 is greater or equal c_2, therefore, it is assumed that SR_1 has at least the same performance as SR_2. On the other hand, if H_0 is not accepted, we assume that SR_1 performs worse than SR_2, on average (EFRON; TIBSHIRANI, 1994). Listing 6 presents an algorithm for finding the confidence intervals using the bootstrap method for PSNR means. For the SSIM metric, the exact same procedure is performed, replacing the PSNR function with the corresponding SSIM function.

Listing 6 Building a confidence interval with the bootstrap method for PSNR means.

```
Input: $\Phi = \{x_1, x_2, ..., x_n\}, x_i \in R$ of PSNR values. The number of bootstrap samples: $B$.
    The confidence error: $\alpha, 0 \leq \alpha \leq 1$
Output: Confidence interval c: $[c_a, c_b]$
1: for b=1:B do
2:     $\Phi^*_b \leftarrow$ resample $\Phi$ with replacement to obtain a data set of size n
3:     $\hat{\theta}^*_b \leftarrow PSNR(\Phi^{*b}_i), \forall i = 1, 2...n$
4:     $m_b \leftarrow$ mean of $\hat{\theta}^*_b$
5: end for
6: Sort the means $m_i, i = 1, 2...B$ in ascending order
7: $a \leftarrow round\Big(B\frac{\alpha}{2}\Big)$
8: $b \leftarrow round\Big(B\Big(1 - \frac{\alpha}{2}\Big)\Big)$
9: return $[m_a, m_b]$
```

5.3 Methodology of the tests

The tests are performed against 4K images with resolution of 3840x2160 pixels. The values for the PSNR and SSIM are taken for both the proposed and selected methods we are comparing to. A widely used methodology for using image quality metrics to compare methods of interest is, considering a magnification factor m_i, to simply downsample the images by m_i and subsequently apply the method being evaluated with the magnification factor m_i. This will result in a reconstructed distorted image $g(x, y)$ that is of the same size as the original reference image $f(x, y)$ and thus f and g can be used to calculate the PSNR and SSIM.

The strategy above provides a simple one-off way of comparing methods, however, in order to run bootstrap to obtain confidence intervals, and therefore obtain a better statistical comparison, we break the images down into several smaller blocks and run the strategy above for each block, instead of the full image. This will result in an array of PSNR and SSIM values that will then be used to build the confidence intervals using the bootstrap method. The idea of running the image quality assessment metrics per image block of arbitrary sizes is also suggest by Wang *et al.* (2004).

In our methodology, the first step is to break the image $g(x, y)$ into several blocks of same size (k_1, k_2), then these blocks are downsampled by the magnification factor m_i and subsequently upsampled by each one of the SR methods being compared. For each SR method and block, the PSNR and SSIM between the reconstructed image block and the reference image block (the image block before being downsampled) are obtained. This generates a set of PSNR and SSIM values for each block of the image, which will in turn, be used to obtain the confidence intervals for each method. Listing 7 summarizes the strategy to build the confidence intervals for both metrics for a given image I.

The proposed method was implemented in the Python language (ROSSUM, 1995)

Listing 7 Building a confidence interval with the bootstrap method for PSNR and SSIM for a given image I.

```
Input: I: image of size (h,w). $SR = \{SR_1, SR_2, ..., SR_n\}$ a set of distinct SR methods.
    $K = (k_1, k_2)$ the size of each block. $m_i$: the magnification factor.
Output: C: set of confidence intervals $C = \{C_1, C_2, ..., C_n\}$, with $C_i = [c_a, c_b]$ for both
    image quality metrics
1: $blocks \leftarrow$ An set of non-overlapping image blocks of size K of the image I
2: for each b of blocks do
3:     $b_d \leftarrow$ downsample b by $m_i$
4:     for each method of SR do
5:         $g \leftarrow SR_i(b_d, m_i)$
6:         $P_{i,b} \leftarrow PSNR(b, g)$
7:         $S_{i,b} \leftarrow SSIM(b, g)$
8:     end for
9: end for
10: for each SR method $SR_i$ do
11:     $C_{i,psnr} \leftarrow$ the psnr confidence interval with $\Phi = P_i$.
12:     $C_{i,ssim} \leftarrow$ the ssim confidence interval with $\Phi = S_i$.
13: end for
```

and the full implementation is listed in Appendix A. The OpenCV framework (BRADSKI, 2000) was used to run the other methods being compared in our test suite. The implementation of the test suite can also be found in Appendix B.

5.4 Results and Discussions

The experimental tests were performed with a subset of the Ultra-Eye UHD and HD image data set (NEMOTO *et al.*, 2014). This data set contains several 4K images of both natural images and historical paintings. The size K of the image blocks used was defined as (64,36). Two magnification factor m_i were applied: 2 and 4, the tests were limited to these factors as the image blocks are not sufficiently big to allow for bigger magnification factors. For building the confidence intervals the following parameters were used $B = 2000$, $\alpha = 0.95$ and the size of the resampled data being the same size of the input data. Some of the tested images are shown in Figure 28.

Even though the proposed method can be extended to be applied for color images, we conducted the tests using grayscale images, therefore all RGB images were converted to the YIQ colospace and only the Y channel was considered when running the tests. The results are summarized in Tables 2 and 3. The best algorithm(s) for each image are highlighted in bold.

Figure 28 – Example images of the Ultra-Eye data set.

(a)

(b)

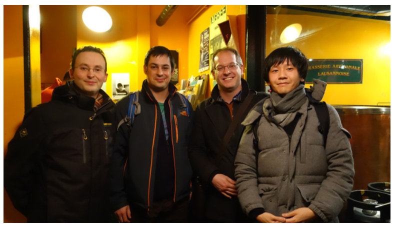
(c)

(d)

(e)

(f)

(g)


(h)

(i)

(j)

(k)

(l)

(m)

(n)

(o)

Source: Nemoto *et al.* (2014).

Table 2 – The confidence intervals for PSNR and SSIM means with zooming factor of 2 and $\alpha = 0.95$.

Image	PSNR				SSIM			
	BCZ	Nearest	Bilinear	Bicubic	BCZ	Nearest	Bilinear	Bicubic
1	**(36.547,36.924)**	(34.742,35.118)	(35.801,36.156)	**(36.864,37.233)**	**(0.94,0.942)**	(0.904,0.906)	(0.919,0.921)	(0.932,0.934)
2	(36.526,37.036)	(34.333,34.813)	(35.786,36.288)	**(37.531,38.019)**	**(0.939,0.941)**	(0.902,0.904)	(0.917,0.92)	**(0.937,0.939)**
3	**(32.89,33.17)**	(31.867,32.154)	(32.192,32.467)	(32.575,32.832)	**(0.898,0.902)**	(0.875,0.878)	(0.879,0.882)	(0.89,0.893)
4	**(32.229,32.447)**	(30.906,31.125)	(31.541,31.76)	**(32.185,32.398)**	**(0.903,0.906)**	(0.874,0.876)	(0.883,0.885)	(0.899,0.901)
5	**(38.596,38.976)**	(37.089,37.448)	(37.738,38.125)	(38.205,38.582)	**(0.922,0.926)**	(0.896,0.9)	(0.9,0.904)	(0.907,0.911)
6	**(33.15,33.631)**	(32.121,32.579)	(32.432,32.903)	**(32.748,33.211)**	**(0.829,0.835)**	(0.823,0.828)	(0.811,0.817)	**(0.83,0.836)**
7	(34.898,35.187)	(32.654,32.932)	(34.173,34.452)	**(35.416,35.679)**	**(0.956,0.959)**	(0.917,0.919)	(0.934,0.937)	(0.944,0.946)
8	**(36.35,36.573)**	(35.076,35.292)	(35.584,35.809)	(35.953,36.163)	**(0.939,0.941)**	(0.907,0.909)	(0.917,0.92)	(0.923,0.925)
9	**(32.949,33.441)**	(31.929,32.397)	(32.247,32.727)	(32.71,33.17)	**(0.889,0.894)**	(0.87,0.874)	(0.87,0.874)	**(0.887,0.891)**
10	**(38.226,38.389)**	(36.871,37.039)	(37.447,37.605)	(37.846,37.988)	**(0.959,0.96)**	(0.923,0.924)	(0.937,0.938)	(0.942,0.943)
11	**(40.666,41.117)**	(38.331,30,350)	(39.829,39,840)	**(41.01,41,02)**	**(0.98,0.981)**	(0.936,0.938)	(0.955,0.956)	(0.96,0.961)
12	(39.817,40.125)	(36.864,37.178)	(39.007,39.319)	**(40.477,40.759)**	**(0.975,0.976)**	(0.932,0.934)	(0.952,0.954)	(0.959,0.96)
13	**(32.432,32.944)**	(31.372,31.862)	(31.786,32.282)	**(32.237,32.74)**	**(0.872,0.877)**	(0.855,0.858)	(0.853,0.858)	(0.873,0.877)
14	**(24.032,24.289)**	(23.472,23.723)	(23.592,23.833)	**(23.938,24.187)**	**(0.701,0.708)**	(0.73,0.735)	(0.695,0.7)	(0.733,0.738)
15	**(37.894,38.448)**	(36.504,37.031)	(37.072,37.601)	**(37.561,38.059)**	**(0.937,0.941)**	(0.904,0.907)	(0.914,0.918)	(0.926,0.929)
16	(28.041,28.261)	(26.908,27.124)	(27.456,27.663)	**(28.299,28.505)**	(0.871,0.874)	(0.85,0.852)	(0.852,0.855)	**(0.877,0.879)**
17	**(36.022,36.58)**	(34.751,35.269)	(35.249,35.77)	(35.693,35.834)	**(0.883,0.889)**	(0.865,0.871)	(0.863,0.869)	(0.875,0.881)
18	**(49.107,49.474)**	(45.651,45.998)	(47.952,48.303)	**(49.412,49.744)**	**(0.992,0.992)**	(0.949,0.95)	(0.966,0.967)	(0.969,0.97)
19	**(35.887,36.201)**	(34.256,34.563)	(35.135,35.446)	**(36.066,36.352)**	**(0.944,0.946)**	(0.909,0.911)	(0.923,0.925)	(0.934,0.936)
20	**(40.72,41.218)**	(39.303,39.775)	(39.823,40.296)	**(40.268,40.650)**	**(0.947,0.95)**	(0.916,0.918)	(0.925,0.928)	(0.932,0.935)

Table 3 – The confidence intervals for PSNR and SSIM means with magnification factor of 4 and $\alpha = 0.95$.

Image	PSNR				SSIM			
	BCZ	Nearest	Bilinear	Bicubic	BCZ	Nearest	Bilinear	Bicubic
1	**(35.29,35.619)**	(32.011,32.357)	(34.619,34.961)	(35.137,35.210)	**(0.935,0.938)**	(0.877,0.881)	(0.915,0.919)	(0.92,0.923)
2	**(32.291,32.611)**	(30.188,30.525)	(31.704,32.027)	(31.698,32.019)	**(0.88,0.885)**	(0.834,0.839)	(0.864,0.868)	(0.867,0.871)
3	**(29.28,29.528)**	(27.264,27.501)	(28.741,28.972)	(28.691,28.927)	**(0.823,0.827)**	(0.772,0.777)	(0.809,0.813)	(0.814,0.818)
4	**(30.693,31.006)**	(27.976,28.253)	(30.157,30.456)	(30.397,30.691)	**(0.9,0.904)**	(0.844,0.848)	(0.883,0.887)	(0.887,0.891)
5	**(30.765,31.319)**	(28.96,29.472)	(30.157,30.663)	(30.009,30.535)	**(0.809,0.816)**	(0.773,0.781)	(0.796,0.803)	(0.8,0.807)
6	**(38.274,38.761)**	(35.965,36.442)	(37.422,37.906)	(37.308,37.812)	**(0.905,0.91)**	(0.86,0.865)	(0.885,0.89)	(0.886,0.891)
7	**(30.004,30.551)**	(28.148,28.695)	(29.41,29.933)	(29.276,29.827)	**(0.773,0.781)**	(0.735,0.743)	(0.761,0.769)	(0.768,0.767)
8	**(22.067,22.321)**	(20.654,20.903)	(21.676,21.935)	(21.493,21.744)	**(0.555,0.565)**	(0.538,0.546)	(0.555,0.564)	**(0.564,0.573)**
9	**(36.089,36.456)**	(33.776,34.148)	(35.264,35.649)	(35.234,35.626)	**(0.882,0.888)**	(0.837,0.843)	(0.861,0.867)	(0.859,0.865)
10	**(36.959,37.471)**	(34.181,34.230)	(36.382,36.453)	(36.622,36,743)	**(0.948,0.95)**	(0.894,0.897)	(0.927,0.93)	(0.93,0.933)
11	**(35.485,36.093)**	(33.407,33.997)	(34.738,35.31)	(34.654,35.23)	**(0.871,0.878)**	(0.829,0.836)	(0.853,0.86)	(0.857,0.863)
12	**(31.691,32.206)**	(29.305,29.813)	(31.192,31.707)	**(31.334,31.816)**	**(0.833,0.838)**	(0.787,0.792)	(0.823,0.827)	**(0.83,0.835)**
13	**(31.004,31.493)**	(29.129,29.604)	(30.303,30.784)	(30.121,30.622)	**(0.731,0.74)**	(0.693,0.702)	(0.718,0.727)	(0.719,0.728)
14	**(30.663,30.964)**	(28.81,29.116)	(30.023,30.32)	(29.854,30.145)	**(0.836,0.842)**	(0.791,0.797)	(0.819,0.825)	(0.82,0.826)
15	**(36.195,36.393)**	(34.027,34.227)	(35.422,35.613)	(35.303,35.491)	**(0.933,0.935)**	(0.885,0.887)	(0.911,0.914)	(0.912,0.914)
16	**(33.607,34.164)**	(31.564,32.098)	(32.906,33.437)	(32.784,33.329)	**(0.817,0.825)**	(0.774,0.783)	(0.8,0.808)	(0.8,0.809)
17	**(24.49,24.722)**	(22.807,23.042)	(24.079,24.319)	(23.973,24.196)	**(0.737,0.742)**	(0.696,0.701)	(0.728,0.734)	**(0.735,0.74)**
18	**(34.301,34.537)**	(32.137,32.362)	(33.565,33.797)	(33.495,33.719)	**(0.908,0.911)**	(0.859,0.862)	(0.887,0.89)	(0.888,0.891)
19	**(32.866,33.295)**	(30.643,31.035)	(32.309,32.694)	(32.411,32.813)	**(0.868,0.872)**	(0.816,0.821)	(0.853,0.857)	(0.859,0.863)
20	**(44.397,44.796)**	(40.74,41.093)	(43.39,43.784)	(43.794,44.211)	**(0.976,0.978)**	(0.929,0.931)	(0.952,0.954)	(0.953,0.954)

Table 4 – The confidence intervals for PSNR and SSIM means with zooming factor of 8 and confidence level of 95%.

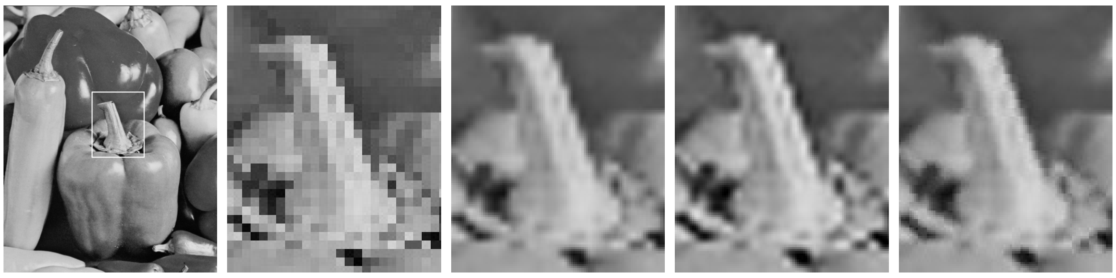
Image	PSNR				SSIM			
	BCZ	Nearest	Bilinear	Bicubic	BCZ	Nearest	Bilinear	Bicubic
1	**(30.946,31.002)**	(28.871,28.916)	(30.284,30.329)	(30.035,30.086)	**(0.717,0.718)**	(0.662,0.663)	(0.705,0.706)	(0.7,0.701)
2	**(28.568,28.615)**	(26.37,26.417)	(28.091,28.138)	(27.85,27.9)	**(0.729,0.73)**	(0.663,0.664)	(0.724,0.725)	(0.72,0.721)
3	**(27.122,27.151)**	(24.708,24.736)	(26.711,26.742)	(26.632,26.665)	**(0.819,0.819)**	(0.747,0.748)	(0.81,0.811)	(0.806,0.807)
4	**(29.225,29.28)**	(27.291,27.345)	(28.665,28.732)	(28.415,28.468)	**(0.705,0.706)**	(0.659,0.66)	(0.696,0.697)	(0.694,0.695)
5	**(34.892,34.954)**	(32.871,32.936)	(34.224,34.28)	(33.962,34.024)	**(0.807,0.808)**	(0.76,0.76)	(0.794,0.794)	(0.788,0.789)
6	**(31.162,31.201)**	(28.598,28.631)	(30.696,30.727)	(30.659,30.695)	**(0.87,0.871)**	(0.799,0.799)	(0.856,0.857)	(0.853,0.854)
7	**(34.059,34.11)**	(30.256,30.301)	(33.472,33.525))	(33.759,33.805)	**(0.892,0.892)**	(0.83,0.83)	(0.876,0.877)	(0.873,0.874)
8	**(21.677,21.704)**	(20.186,20.212)	(21.279,21.301)	(20.984,21.006)	**(0.504,0.505)**	(0.462,0.463)	(0.503,0.504)	(0.5,0.5)
9	**(30.442,30.494)**	(28.377,28.415)	(29.936,29.976)	(29.703,29.748)	**(0.791,0.792)**	(0.726,0.726)	(0.782,0.783)	(0.779,0.78)
10	**(37.063,37.11)**	(34.676,34.724)	(36.267,36.319)	(36.061,36.102)	**(0.874,0.875)**	(0.821,0.822)	(0.856,0.857)	(0.853,0.853)
11	**(22.685,22.709)**	(21.044,21.069)	(22.356,22.383)	(22.036,22.059)	**(0.634,0.635)**	(0.577,0.578)	(0.633,0.633)	(0.628,0.628)
12	**(33.093,33.116)**	(30.824,30.845)	(32.44,32.462)	(32.275,32.298)	**(0.892,0.892)**	(0.832,0.832)	(0.873,0.874)	(0.872,0.872)
13	**(35.127,35.158)**	(32.757,32.793)	(34.438,34.475)	(34.26,34.295)	**(0.886,0.886)**	(0.831,0.831)	(0.867,0.867)	(0.864,0.864)
14	**(30.46,30.514)**	(28.578,28.632)	(29.87,29.914)	(29.619,29.678)	**(0.75,0.751)**	(0.705,0.706)	(0.742,0.743)	(0.737,0.738)
15	**(27.18,27.203)**	(25.239,25.263)	(26.735,26.76)	(26.514,26.537)	**(0.755,0.756)**	(0.689,0.69)	(0.746,0.746)	(0.744,0.744)
16	**(33.064,33.127)**	(30.947,31.002)	(32.422,32.475)	(32.164,32.223)	**(0.794,0.795)**	(0.744,0.745)	(0.78,0.781)	(0.776,0.777)
17	**(30.102,30.133)**	(28.073,28.11)	(29.566,29.598)	(29.284,29.311)	**(0.821,0.821)**	(0.759,0.76)	(0.807,0.807)	(0.801,0.801)
18	**(40.251,40.29)**	(37.199,37.238)	(39.517,39.554)	(39.529,39.566)	**(0.955,0.956)**	(0.902,0.903)	(0.935,0.935)	(0.933,0.934)
19	**(35.34,35.361)**	(32.983,33.005)	(34.601,34.625)	(34.421,34.446)	**(0.918,0.918)**	(0.858,0.859)	(0.898,0.899)	(0.897,0.897)
20	**(29.931,29.968)**	(28.025,28.056)	(29.368,29.403)	(29.087,29.118)	**(0.804,0.805)**	(0.749,0.75)	(0.789,0.79)	(0.786,0.787)

For smaller factors of zoom (Table 2), the experimental results indicates, according to the PSNR metric, that our method has the same performance, most of the time, as the bicubic interpolation. In six of the tested images, our method outperformed all other methods, including the bicubic method. On the other hand, according to the SSIM metric, which is more appropriate for assessing the perceived visual quality, our method has outperformed all the other ones in almost every image. The bicubic algorithm performed better or equal to our method in only four images.

We have also found that, for bigger factors of zoom, our method tends to perform better as we can see in Table 3, in which the proposed method did not perform worse than any other method in any of the tested image. This trend is confirmed by the experiments carried out with a zooming factor of 8 as seen in Table 4, where BCZ outperformed the other methods in all images.

Figure 29 shows a region of the classic "peppers image", zoomed by each method. It is easy to observe severe aliasing artifacts in the image augmented by the nearest interpolation method. The zoomed image of the bilinear method has less aliasing artifacts, but the image in general became over-smoothed, which in turn, diminishes the sharpness of the image. The bicubic method, as expected, performed better than the other classic methods. However, the bilinear and bicubic methods have the ringing effect, which is the rippling artifacts near the edges. The BCZ method, on the other hand, produced a high-resolution image with less aliasing artifacts and no ringing effect. In Figure 30, we performed a zoom in a region of the "camera man" image. With the exception of the nearest interpolation, the methods have performed slightly similar, however we can see that BCZ is preserving the transition from the black jacket to the gray background a bit better, without adding the "ringing" effect as observed in the image generated by Bicubic interpolation.

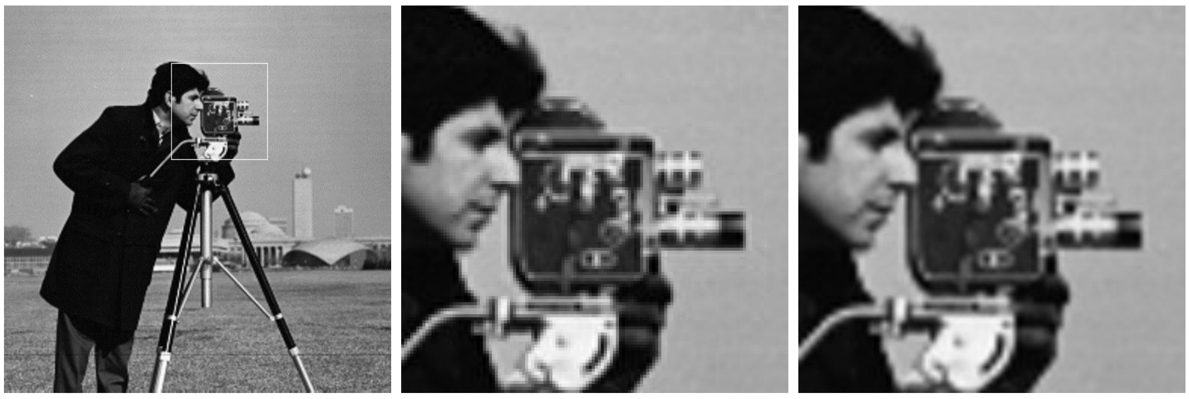
Figure 29 – A comparison of a region of the classic "peppers" image zoomed by each of the evaluated methods. In (a)zoomed 4x with nearest Interpolation. In (b) zoomed 4x with bilinear interpolation. In (c) zoomed 4x with cubic interpolation. In (d) zoomed 4x with BCZ interpolation.

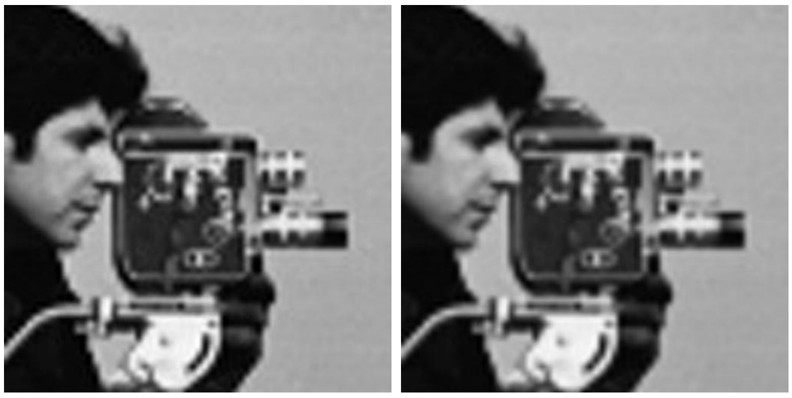


(a) (b) Nearest (c) Bilinear (d) Bicubic (e) BCZ

Source: Adapted from Gonzalez e Woods (2006).

Figure 30 – A comparison of a region of the classic "camera man" image zoomed by each of the evaluated methods. In (a)zoomed 4x with nearest Interpolation. In (b) zoomed 4x with bilinear interpolation. In (c) zoomed 4x with cubic interpolation. In (d) zoomed 4x with BCZ interpolation.


(a) (b) Nearest (c) Bilinear


(d) Bicubic (e) BCZ

Source: Own authorship.

6 Conclusion and Future Work

In this thesis, we have proposed a new non-linear and adaptive image super-resolution algorithm called BCZ that interpolates the pixels using the multiple linear regression model as expressed in Equation (3.6). This linear regression model is used alongside with the Box-Cox transformation to effectively obtain the non-linear and adaptive behavior of the proposed method. Box-Cox was used to linearize the pixel distribution prior to running the regression with the goal of yielding better estimates for the unknown pixels of the high-resolution image. In Chapter 4, the method was described in detail, in which, the basic strategy of using multiple linear regression as a interpolation function was shown and how Box-Cox was introduced to improve the method, reducing the error of the estimated pixels.

Our proposed method was compared to three of the classical interpolation-based super-resolution algorithms: nearest interpolation, bilinear interpolation and bicubic interpolation. They are simple, fast and widely used algorithms. Suffice to say, all of the major image editing software include these methods. For the average user, these methods provide a reasonable performance for most of the day-to-day needs. However, for other application such as obtaining high-resolution images from surveillance cameras, the need for a method that preserves the sharpness and fine details of the images is crucial.

Two image quality assessment metrics were used to evaluate the proposed method and compare it to the classical methods: PSNR and SSIM. The first is a classic metric that basically computes the average of the squared intensity differences of the degraded and reference image pixels. The SSIM, on the other hand, is an attempt to better model the human visual system (HVS) and express it in a mathematical expression as seen in Equation (5.10). The SSIM metric measures the structural similarity of two images. In theory, it is a much better choice for measuring the perceived visual quality of images.

For a better statistical comparison between the evaluated methods, the bootstrap method was used to build confidence intervals. A special procedure used to conduct the tests was described in Chapter 5, in which, the image is broken down into several blocks of same size and the metrics are executed against the image blocks, instead of the full image. Twenty images from the Ultra Eye (NEMOTO *et al.*, 2014) image data set were used. The results of the tests are summarized in Tables 2 and 3. We can conclude from the experimental tests that our method outperforms, on average, the classical methods.

The ultimate goal of the proposed method was to overcome the inherent limitation of the classical methods, which tends to attenuate the high frequencies of the resulting images, making them feel over-smoothed, losing the sharpness and the fine details. The proposed BCZ method proved to outperform all of the classical methods, tending to perform better with higher zooming factor (zooming factor higher than 2), where most of the SR methods do not perform well. High-resolution images generated by BCZ are sharp and free of the most common artifacts such as ringing and aliasing.

6.1 Future Work

A big downside of the proposed method is the complexity of it. Running a regression model for every pixel is inherently a computationally expensive task. The strategy for finding the best lambda, while extremely better compared to the naive approach, is still expensive, given it is executed for every new pixel in the high-resolution image. The classical methods would take milliseconds to augment an standard high definition (HD) image, while the proposed method usually takes tens of seconds, making it not suitable for real time tasks. However, for non real time tasks, the proposed method might be a good candidate. With that in mind, several optimization improvements can be made to the existing implementation as future work, including parallelization and the use of GPU (Graphics Processing Units) for running expensive calculations.

The method described in this thesis has the potential to be extended and improved in order to be used in image restoration problems (TAKAHASHI; ISHII, 1991), which is another suggestion for future work. The input for an image restoration problem is an image $I(x, y)$ with pixels missing or damaged by an unknown process. The goal of image restoration is to reconstruct a image $g(x, y)$ from the damaged image $I(x, y)$, that is a good approximation of the original unknown image.

Bibliography

BAKER, S.; KANADE, T. Hallucinating faces. In: *Proceedings Fourth IEEE International Conference on Automatic Face and Gesture Recognition (Cat. No. PR00580)*. [S.l.: s.n.], 2000. p. 83–88. Cited at page 25.

BAKER, S.; KANADE, T. Limits on super-resolution and how to break them. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 24, n. 9, p. 1167–1183, Sept 2002. ISSN 0162-8828. Cited 2 time at pages 14 and 25.

BEN-ISRAEL, A. A newton-raphson method for the solution of systems of equations. *Journal of Mathematical analysis and applications*, Academic Press, v. 15, n. 2, p. 243–252, 1966. Cited at page 44.

BERNSTEIN, D. *Matrix Mathematics: Theory, Facts, and Formulas with Application to Linear Systems Theory*. Princeton University Press, 2005. ISBN 9780691118024. Disponível em: <https://books.google.com.br/books?id=pmNRPwOFHKoC>. Cited 2 time at pages 43 and 44.

Bocsika. *Bilinear interpolation*. 2009. Disponível em: <https://commons.wikimedia.org/wiki/File:BilinearInterpolation.svg>. Cited at page 28.

BOX, G. E. P.; COX, D. R. An analysis of transformations. *Journal of the Royal Statistical Society. Series B (Methodological*, p. 211–252, 1964. Cited at page 37.

BRADSKI, G. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000. Cited at page 53.

BURGER, W.; BURGE, M. J. *Undergraduate Topics in Computer Science*. [s.n.], 2009. ISSN 1848001959. ISBN 978-1-84800-194-7. Disponível em: <http://www.springerlink.com/index/10.1007/978-1-84800-195-4>. Cited at page 38.

CHATTERJEE, S.; HADI, A. S. *Regression Analysis by Example*. [S.l.]: John Wiley, Sons, Inc., 2006. ISBN 9780470055465. Cited at page 33.

CUI, Z. *et al.* Deep network cascade for image super-resolution. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v. 8693 LNCS, n. PART 5, p. 49–64, 2014. ISSN 16113349. Cited at page 26.

DANIEL, C.; WOOD, F. S. *Fitting Equations to Data: Computer Analysis of Multifactor Data*. 2nd. ed. New York, NY, USA: John Wiley & Sons, Inc., 1980. ISBN 0471053708. Cited at page 35.

DONG, C. *et al.* Learning a deep convolutional network for image super-resolution. In: FLEET, D. *et al.* (Ed.). *Computer Vision – ECCV 2014*. Cham: Springer International Publishing, 2014. p. 184–199. ISBN 978-3-319-10593-2. Cited at page 25.

DOURADO, W. B. Avaliação de técnicas de interpolação de imagens digitais. Universidade Estadual Paulista (UNESP), 2014. Cited 2 time at pages 47 and 50.

EFRON, B.; TIBSHIRANI, R. Bootstrap methods for standard errors, confidence intervals, and other measures of statistical accuracy. *Statist. Sci.*, The Institute of Mathematical Statistics, v. 1, n. 1, p. 54–75, 02 1986. Disponível em: <https://doi.org/10.1214/ss/1177013815>. Cited at page 50.

EFRON, B.; TIBSHIRANI, R. *An Introduction to the Bootstrap*. Taylor & Francis, 1994. (Chapman & Hall/CRC Monographs on Statistics & Applied Probability). ISBN 9780412042317. Disponível em: <https://books.google.com.br/books?id=gLlpIUxRntoC>. Cited 2 time at pages 50 and 51.

FARSIU, S. *et al.* Fast and robust multiframe super resolution. *IEEE Transactions on Image Processing*, v. 13, n. 10, p. 1327–1344, Oct 2004. ISSN 1057-7149. Cited at page 25.

FLORIN, T.; ARSINTE, R.; MASTORAKIS, N. E. Resolution analysis of an image acquisition system. In: *Proceedings of the 5th European Conference on European Computing Conference*. Stevens Point, Wisconsin, USA: World Scientific and Engineering Academy and Society (WSEAS), 2011. (ECC'11), p. 382–391. ISBN 978-960-474-297-4. Disponível em: <http://dl.acm.org/citation.cfm?id=1991016.1991085>. Cited at page 14.

FLORIN, T.; ARSINTE, R.; MASTORAKIS, N. E. Resolution analysis of an image acquisition system. In: *Proceedings of the 5th European Conference on European Computing Conference*. Stevens Point, Wisconsin, USA: World Scientific and Engineering Academy and Society (WSEAS), 2011. (ECC'11), p. 382–391. ISBN 978-960-474-297-4. Disponível em: <http://dl.acm.org/citation.cfm?id=1991016.1991085>. Cited at page 18.

FREEMAN, W. T.; PASZTOR, E. C. Learning low-level vision. In: *Proceedings of the Seventh IEEE International Conference on Computer Vision*. [S.l.: s.n.], 1999. v. 2, p. 1182–1189 vol.2. Cited 2 time at pages 15 and 25.

GEER, S. A. van de. Least Squares Estimation. *Encyclopedia of Statistics in Behavioral Science*, v. 2, p. 1041–1045, 2005. ISSN 1525-8955. Disponível em: <http://doi.wiley.com/10.1002/0470013192.bsa199>. Cited at page 31.

GIROD, B. Digital images and human vision. In: WATSON, A. B. (Ed.). Cambridge, MA, USA: MIT Press, 1993. cap. What's Wrong with Mean-squared Error?, p. 207–220. ISBN 0-262-23171-9. Disponível em: <http://dl.acm.org/citation.cfm?id=197765.197784>. Cited at page 47.

GLASNER, D.; BAGON, S.; IRANI, M. Super-resolution from a single image. *Proceedings of the IEEE International Conference on Computer Vision*, n. Iccv, p. 349–356, 2009. ISSN 1550-5499. Cited 2 time at pages 14 and 25.

GONZALEZ, R. C.; WOODS, R. E. *Digital Image Processing (3rd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006. ISBN 013168728X. Cited 7 time at pages 14, 15, 18, 22, 25, 27, and 58.

HORE, A.; ZIOU, D. Image quality metrics: Psnr vs. ssim. In: *2010 20th International Conference on Pattern Recognition*. [S.l.: s.n.], 2010. p. 2366–2369. ISSN 1051-4651. Cited at page 46.

HUNT, R. *The Reproduction of Colour (6rd Edition)*. [S.l.]: John Wiley, Sons, Ltd, 2005. ISBN 9780470024270. Cited at page 20.

IRANI, M.; PELEG, S. Improving resolution by image registration. *CVGIP: Graph. Models Image Process.*, Academic Press, Inc., Orlando, FL, USA, v. 53, n. 3, p. 231–239, abr. 1991. ISSN 1049-9652. Disponível em: <http://dx.doi.org/10.1016/1049-9652(91)90045-L>. Cited at page 25.

JAMES, G. *et al. An Introduction to Statistical Learning: With Applications in R*. [S.l.]: Springer Publishing Company, Incorporated, 2014. ISBN 1461471370, 9781461471370. Cited at page 49.

KIM, H.; CHA, Y.; KIM, S. Curvature interpolation method for image zooming. *IEEE Transactions on Image Processing*, v. 20, n. 7, p. 1895–1903, 2011. ISSN 10577149. Cited at page 25.

KUPPERMAN, M. Linear Statistical Inference and Its Applications 2nd Edition (C. Radhakrishna Rao). *SIAM Review*, v. 17, n. 4, p. 705–707, 1975. Disponível em: <https://doi.org/10.1137/1017090>. Cited at page 44.

LAHOULOU, A. *et al.* A complete statistical evaluation of state-of-the-art image quality measures. In: *International Workshop on Systems, Signal Processing and their Applications, WOSSPA*. [S.l.: s.n.], 2011. p. 219–222. Cited at page 46.

LEE, H. S.; LEE, K. M. Simultaneous super-resolution of depth and images using a single camera. In: *2013 IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2013. p. 281–288. ISSN 1063-6919. Cited at page 24.

LI, Z.-N.; DREW, M. S.; LIU, J. *Fundamentals of Multimedia*. 2nd. ed. [S.l.]: Springer Publishing Company, Incorporated, 2014. ISBN 3319052896, 9783319052892. Cited 2 time at pages 19 and 21.

LIN, Z.; SHUM, H.-Y. Fundamental limits of reconstruction-based superresolution algorithms under local translation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 26, n. 1, p. 83–97, Jan 2004. ISSN 0162-8828. Cited at page 25.

MAGNUS, J. R.; NEUDECKER, H. *Matrix Differential Calculus with Applications in Statistics and Econometrics*. Second. [S.l.]: John Wiley, 1999. ISBN 0471986321 9780471986324 047198633X 9780471986331. Cited at page 45.

MARTINEZ, W.; MARTINEZ, A. *Computational Statistics Handbook with MATLAB*. CRC Press, 2001. (Chapman & Hall/CRC Computer Science & Data Analysis). ISBN 9781420035636. Disponível em: <https://books.google.com.br/books?id=my-i9VNzCfAC>. Cited at page 50.

MONTGOMERY, D.; PECK, E.; VINING, G. *Introduction to Linear Regression Analysis*. Wiley, 2012. (Wiley Series in Probability and Statistics). ISBN 9780470542811. Disponível em: <https://books.google.com.br/books?id=0yR4KUL4VDkC>. Cited 5 time at pages 15, 30, 33, 34, and 36.

NELSON, E. C. Principles of numerical analysis. alston s. householder. mcgraw-hill, new york-london, 1953. 274 pp. $6. *Science*, American Association for the Advancement of Science, v. 119, n. 3095, p. 547–548, 1954. ISSN 0036-8075. Disponível em: <http://science.sciencemag.org/content/119/3095/547>. Cited at page 44.

NEMOTO, H. *et al.* Ultra-Eye: UHD and HD images eye tracking dataset. 2014. Cited 3 time at pages 53, 54, and 60.

PEDRINI, H.; SCHWARTZ, W. *Análise de imagens digitais: princípios, algoritmos e aplicações*. THOMSON PIONEIRA, 2008. ISBN 9788522105953. Disponível em: <https://books.google.com.br/books?id=13KAPgAACAAJ>. Cited at page 27.

RAWLINGS, J.; PANTULA, S.; DICKEY, D. *Applied Regression Analysis: A Research Tool*. Springer New York, 2001. (Springer Texts in Statistics). ISBN 9780387984544. Disponível em: <https://books.google.com.br/books?id=PMeJGeXA09EC>. Cited 2 time at pages 16 and 44.

ROSSUM, G. *Python Reference Manual*. Amsterdam, The Netherlands, The Netherlands, 1995. Cited at page 52.

ROSZKOWIAK, L. *et al.* Survey: interpolation methods for whole slide image processing. *Journal of Microscopy*, v. 265, n. 2, p. 148–158, 2016. ISSN 00222720. Disponível em: <http://doi.wiley.com/10.1111/jmi.12477>. Cited at page 25.

SAKIA, R. The box-cox transformation technique: a review. *The statistician*, JSTOR, p. 169–178, 1992. Cited at page 15.

SCHUON, S. *et al.* Lidarboost: Depth superresolution for tof 3d shape scanning. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2009. p. 343–350. ISSN 1063-6919. Cited at page 24.

STYAN, G. P. H. Hadamard products and multivariate statistical analysis. *Linear Algebra and its Applications*, v. 6, p. 217–240, 1973. ISSN 0024-3795. Disponível em: <http://www.sciencedirect.com/science/article/pii/0024379573900232>. Cited at page 45.

TAKAHASHI, K.; ISHII, N. Restoration of images with missing pixels. *Systems and Computers in Japan*, v. 22, n. 2, p. 34–41, 1991. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/scj.4690220205>. Cited at page 61.

TEO, P. C.; HEEGER, D. J. Perceptual image distortion. In: *Proceedings of 1st International Conference on Image Processing*. [S.l.: s.n.], 1994. v. 2, p. 982–986 vol.2. Cited at page 47.

THYAGARAJAN, K. *Still Image and Video Compression with MATLAB*. Wiley, 2011. ISBN 9781118097762. Disponível em: <https://books.google.com.br/books?id=adv71MkRIWYC>. Cited at page 18.

Todd Veldhuizen. *Measures of image quality*. 1998. [Online; acessado em 02 de Outubro de 2016]. Disponível em: <http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/VELDHUIZEN/node18.html>. Cited at page 48.

UNSER, M.; ALDROUBI, A.; EDEN, M. Fast b-spline transforms for continuous image representation and interpolation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 13, n. 3, p. 277–285, Mar 1991. ISSN 0162-8828. Cited at page 25.

WANG, Z.; BOVIK, A. C. Mean squared error: Love it or leave it? a new look at signal fidelity measures. *IEEE Signal Processing Magazine*, v. 26, n. 1, p. 98–117, Jan 2009. ISSN 1053-5888. Cited at page 50.

WANG, Z.; BOVIK, A. C.; LU, L. Why is image quality assessment so difficult? In: *2002 IEEE International Conference on Acoustics, Speech, and Signal Processing.* [S.l.: s.n.], 2002. v. 4, p. IV–3313–IV–3316. ISSN 1520-6149. Cited at page 47.

WANG, Z. *et al.* Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, v. 13, n. 4, p. 600–612, April 2004. ISSN 1057-7149. Cited 4 time at pages 47, 49, 50, and 52.

WYSZECKI, G.; STILES, W. *Color Science: Concepts and Methods, Quantitative Data and Formulae.* Wiley, 2000. (Wiley Series in Pure and Applied Optics). ISBN 9780471399186. Disponível em: <https://books.google.com/books?id=_51HDcjWZPwC>. Cited at page 20.

XIAN, Y.; TIAN, Y. Single image super-resolution via internal gradient similarity. *Journal of Visual Communication and Image Representation*, Elsevier Inc., v. 35, p. 91–102, 2016. ISSN 10959076. Disponível em: <http://dx.doi.org/10.1016/j.jvcir.2015.11.015>. Cited at page 25.

YANG, J.; LIN, Z.; COHEN, S. Fast image super-resolution based on in-place example regression. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, p. 1059–1066, 2013. ISSN 10636919. Cited at page 14.

ZHANG, W. *et al.* Wavelet Transform and High-lever PDE for Image Zooming. *Proceedings - 2015 8th International Symposium on Computational Intelligence and Design, ISCID 2015*, v. 2, p. 212–214, 2016. Cited at page 25.

Appendix

APPENDIX A – Proposed method implementation

Listing A.1 – The proposed method

```
import numpy as np
from scipy.stats import boxcox
from scipy.special import inv_boxcox
from tqdm import tqdm

import math

def get_window(img, x,y):
    coords = np.array([[x,y],[x+1,y],[x,y+1],[x+1,y+1]])
    Y = np.array([img[x,y] for x,y in coords ])

    x_vector = np.array(coords[:,0])
    y_vector = np.array(coords[:,1])

    N = coords.shape[0]

    return x_vector, y_vector, Y, N

def regress(X,Y):
    XT = np.transpose(X)
    XTX = np.dot(XT,X)

    b = np.dot( np.dot( np.linalg.inv( XTX ), XT ), Y )

    return b

def get_pixel(sx,sy, regression_line):
    return regression_line[0] + regression_line[1] * sx + regression_line[2] * sy

def find_lambda(X,y):
    XT = np.transpose(X)
    XTX = np.dot(XT,X)
    H = np.dot( np.dot(X, np.linalg.inv( XTX ) ), XT)
    H = np.min(H) + H
    IH = np.eye(H.shape[0])-H
    _lambda = 1

    for i in range(5):
        y_lambda = np.power(y, _lambda)
        y_lambda_1 = y_lambda - 1
        S = np.dot(np.dot( np.transpose(y_lambda_1), IH ), y_lambda_1) / (_lambda**2)
        S_ = (np.dot( np.dot(np.transpose(2*(y_lambda) * np.log(y)), IH), y_lambda_1) - 2 *_lambda * S
        fx = S*_lambda - np.dot( np.dot( np.transpose(y_lambda*np.log(y)), IH), y_lambda_1)
        fx_1 = S_ * _lambda + S - np.dot( np.dot( np.transpose( y_lambda * np.power(np.log(y), 2) ), IH
        fx_2 = np.dot( np.dot( np.transpose( y_lambda * np.log(y)), IH ), y_lambda * np.log(y) )
        f_x = fx_1 - fx_2
        new_lambda = _lambda - fx/f_x

        if abs( _lambda-new_lambda) < 1e-4 : #checar parametro de erro
            break

        _lambda = new_lambda


    return _lambda

def zoom(img, factor):
    h, w = img.shape
    h2, w2 = h * factor, w * factor
    hr_image = np.zeros((h2,w2), dtype=np.float64)
    x_ratio,y_ratio = w / w2, h / h2
    sx = sy = 0

    for i in tqdm(range(h2)):
        for j in range(w2):
            sx = (i + 0.5) * x_ratio - 0.5
            sy = (j + 0.5) * y_ratio - 0.5

            sx = 0 if sx < 0 else sx
            sy = 0 if sy < 0 else sy

            dx, dy = int(sx), int(sy)
            dx = dx-1 if dx == h-1 else dx
            dy = dy-1 if dy == w-1 else dy

            x_vector, y_vector, Y, N = get_window(img, dx, dy)
            X = np.ones( (N,3) )
            X[:,1] = x_vector
            X[:,2] = y_vector

            Y = Y + 1
            lambda_val = find_lambda(X, Y)

            YT = boxcox(Y,lambda_val)
            regression_line = regress(X,YT)
            pixel = get_pixel(sx,sy, regression_line)
            pixel = inv_boxcox(pixel, lambda_val)
            pixel = pixel −1

            hr_image[i,j] = pixel

    return hr_image
```

Listing A.2 – Example usage of the proposed method

```
import srlr
import cv2
from skimage.io import imread
from skimage.transform import rescale
from matplotlib import pyplot as plt
from skimage.measure import compare_ssim as ssim
from skimage.measure import compare_psnr as psnr
import numpy as np

factor = 2
original_image = imread('images/C01_HD.tif', True)
img = rescale(original_image, 1/factor, multichannel=False, anti_aliasing=False )

srlr_hr = srlr.zoom(img, factor)
neighborhood_hr = cv2.resize(img, original_image.shape[::−1], interpolation=cv2.INTER_NEAREST)
bilinear_hr = cv2.resize(img, original_image.shape[::−1])

plt.imshow(srlr_hr, cmap='gray')
plt.show()

print(psnr(original_image,srlr_hr))
print(psnr(original_image,neighborhood_hr))
print(psnr(original_image,bilinear_hr))
```

APPENDIX B – Test suite implementation

Listing B.1 – Code that obtains the PSNR and SSIM metric for each image block

```
import srlr
import cv2
import pandas as pd
from skimage.io import imread
from matplotlib import pyplot as plt
from skimage.transform import rescale
from skimage.measure import compare_ssim as ssim
from skimage.measure import compare_psnr as psnr
from skimage.util.shape import view_as_blocks as getBlocks
from tqdm import tqdm
import numpy as np
import os

factors = [4, 6, 8, 16]
K =(36,64)

for factor in factors:
    for image_i in range(1,21):
        image = 'images/4kimages/C' + str(image_i) + '_HD.tif'
        basename = os.path.basename(image)
        print('Running ' + str(factor) + 'x for ' + basename)

        psnr_blocks = []
        ssim_blocks = []
        neighborhood_psnr_blocks = []
        neighborhood_ssim_blocks = []
        bilinear_psnr_blocks = []
        bilinear_ssim_blocks = []
        bicubic_psnr_blocks = []
        bicubic_ssim_blocks = []

        original_image = imread(image, True)
        blocks = getBlocks(original_image, block_shape=K)
        bbar = tqdm(total=len(blocks))
        for bi in blocks:
            for b in bi:
```

```
            bd = rescale(b, 1/factor, multichannel=False, anti_aliasing=False )

            srlr_hr = srlr.zoom(bd, factor)
            psnr_blocks.append(psnr(b,srlr_hr))
            ssim_blocks.append(ssim(b,srlr_hr))

            neighborhood_hr = cv2.resize(bd, K, interpolation=cv2.INTER_NEAREST).reshape(K)
            neighborhood_psnr_blocks.append(psnr(b,neighborhood_hr))
            neighborhood_ssim_blocks.append(ssim(b,neighborhood_hr))

            bilinear_hr = cv2.resize(bd, K).reshape(K)
            bilinear_psnr_blocks.append(psnr(b,bilinear_hr))
            bilinear_ssim_blocks.append(ssim(b,bilinear_hr))

            bicubic_hr = cv2.resize(bd, K, interpolation=cv2.INTER_CUBIC).reshape(K)
            bicubic_psnr_blocks.append(psnr(b,bicubic_hr))
            bicubic_ssim_blocks.append(ssim(b,bicubic_hr))
        bbar.update(1)

    df = pd.DataFrame( {
        'srlr_psnr': psnr_blocks, 'srlr_ssim': ssim_blocks,
        'neighborhood_psnr': neighborhood_psnr_blocks, 'neighborhood_ssim': neighborhood_ssim_block
        'bilinear_psnr': bilinear_psnr_blocks, 'bilinear_ssim': bilinear_ssim_blocks,
        'bicubic_psnr': bicubic_psnr_blocks, 'bicubic_ssim': bicubic_ssim_blocks
    } )

    df.to_csv('results/' + os.path.splitext(basename)[0] + '-' + str(factor) + 'x.csv', index=False)
```

Listing B.2 – The confidence interval function

```
from sklearn.utils import resample
import numpy as np

def confidence_interval(data, B, alpha, n_samples = 500):
    m = []
    for i in range(B):
        new_sample = resample(data, n_samples=n_samples)
        m.append(np.mean(new_sample))
    m.sort()

    a = round(B*(alpha/2))
    b = round(B*(1 - alpha/2))

    return (m[a],m[b])
```

Listing B.3 – Code that obtains the confidence intervals for each method

```
import glob
import os
import pandas as pd
from utils import confidence_interval

for filepath in glob.iglob('results/*.csv'):
    basename = os.path.basename(filepath)
    df = pd.read_csv(filepath)
    print(basename)
    for column in df:
        print(column, confidence_interval(df[column], 2000, 0.99, 900))
```