IMPLEMENTATION OF CAPTCHA PATTERN RECOGNITION USING DENOISING AND OPTICAL CHARACTER RECOGNITION (OCR) METHODS

Presented by:
HAMILTON SMITH GÓMEZ OSORIO
JULIANA LALINDE VELÁSQUEZ
VERÓNICA MENDOZA IGUARÁN
PABLO ALBERTO OSORIO MARULANDA
MARIANA URIBE ORREGO

Subject
NUMERICAL ANALYSIS

Teacher
EDWAR SAMIR POSADA MURILLO

MATHEMATICAL ENGINEERING
DEPARMENT OF MATHEMATICAL SCIENCES
SCHOOL OF SCIENCES
EAFIT UNIVERSITY
MEDELLÍN
2020

1 INTRODUCTION

The security of websites is of great importance today, as they must protect the information of their users. A CAPTCHA is a test that allows to determine if a user is a human or a robot, it consists of presenting an image with distorted text and the user must recognize said text. If it is a robot it will not be able to recognize it and the test will detect it. These texts are increasingly difficult to recognize, since most of them come with noises which distort the image. Currently there are multiple ways to reduce or eliminate noise in images, so in this research we discuss and compare different methods for noise reduction in order to understand each method in depth and achieve image text recognition. The methods to compare are Median filter, Mean filter, Bilateral filter, Gaussian filter and Wavelet transform. On the other hand we talk about Optical Character Recognition (OCR), which recognizes characters individually in an image and then passes it to a text and thus detects the text it treats.

Our product will be an exhaustive analysis of image processing, in which by means of different methods we will add certain noises to CAPTCHA in order to evaluate which of the studied methods best serve to reduce noise. The comparison is made by interpreting the correlation statistic. The maximization of the correlation coefficient is sought, which is determined by analyzing the set of pixels between the two images. Also, we evaluate which of these processed CAPTCHAs are recognizable by OCR Methods.

2 DENOISING FILTERS

2.1 MEDIAN AND AVERAGE FILTER

The median and average filter work in similar ways, so they are often compared. In this subsection we will explain how both of these filters work and compare their results to see which one works better.

2.1.1 Median filter

The median filter is a non-linear method used to reduce noise from images. It is particularly good at removing "salt and pepper" type noise and preserving the edges of the image intact. It works by moving through the image pixel by pixel and replacing each value with the median value of the neighboring pixels; the number of pixels considered in every step is called the window. Seeing as the windows need to be sorted in every iteration the method can have a high degree of computational complexity if it is not applied properly. Seeing as the window may contain more pixels than there exist before the first pixel the edges of the image are usually excluded from the method.

```
Median filter algorithm
Input: image, window size
Process:
For x from 0 to image length
     For y from 0 to image length
          index = 0
          For fx from 0 to window size
               For fy from 0 to window size
                    window[index] = imagePixel(x+fx,y+fy)
                    index = index + 1
               end For
          end For
          Sort window
          median = window [ (window size)^2 / 2]
          imagePixel(x,y) = median
     end For
end For
```

Figure 1: Pseudo code for the median filter

2.1.2 Average filter

The average or mean filter is a non-linear method used to reduce noise from images. It is particularly good at removing the intensity variation between neighboring pixels, but has trouble preserving the edges of the image intact. It works by moving through the image pixel by pixel and replacing each value with the average value of the neighboring pixels; like the median filter, the number of pixels considered in every step is called the window. This method can be inaccurate if you have a single pixel with a very unrepresentative value can in the image, seeing as it can affect the average value of all the pixels in its neighborhood.

```
Average filter algorithm
Input: image, window size
Process:
For x from 0 to image length
     For y from 0 to image length
          index = 0
          For fx from 0 to window size
               For fy from 0 to window size
                    window[index] = imagePixel(x+fx,y+fy)
                    index = index + 1
               end For
          end For
          Sort window
          average = sum(window) / index
          imagePixel(x,y) = average
     end For
end For
```

Figure 2: Pseudo code for the average filter

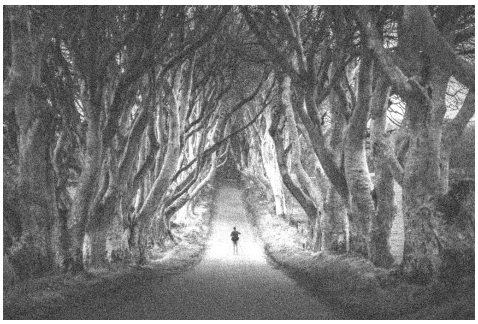
2.1.3 Matlab implementation

To visualize the behaviour of the filters we tested them using Matlab functions. We first took a normal image and added two different noises: some salt and pepper noise, choosing a noise density of 0.15, and some gaussian noise, with a 0.25 parameter.

(a) Original image

(b) Image with salt and pepper noise

(c) Image with gaussian noise



For the median filter we used the Matlab function $medfilt2(I, [m\ n])$, which applies the median filter to a 2D given image. In this function $[m\ n]$ represent the size of the window that will be used. Here is the image we got using the median filter with different window sizes for the salt and pepper noise:

Figure 4: Filtered image with 3x3 window

The best image we got using the median filter for the gaussian noise with the different window size is:

Figure 5: Filtered image with 5x5 window

For the average filter we used the Matlab function *conv*2, which applies a personalized convolution to a 2D image. In this case the filter we gave it to represent the average filter was $ones(windowsize)/(windowsize)^2$. The best image we got using the mean filter with different window sizes for the salt and pepper noise is:

Figure 6: Filtered image with 7x7 window

The best image we got using the median filter for the gaussian noise with the different window size is:

Figure 7: Filtered image with 3x3 window

Then we proceeded to check the correlations between the original image and the filtered images. As seen in the table below, the median filter is more effective in removing the salt and pepper noise added to the images. We can see in the images that the median filter tends to take better care of the edges of the image and keep the shapes clear in general. When it comes to gaussian noise the mean filter seems to have a better correlation level with the original, but when you look at the images neither of the methos seems to be effectively removing this type of noise. For this reason we can conclude that the median filter has an overral better performance removind salt and pepper noise and neither of the filters seems to do a good job with the gaussian noise.

	3x3		5x5		7x7	
	Mean filter	**Median filter**	**Mean filter**	**Median filter**	**Mean filter**	**Median filter**
GAUSS	Filtered	Filtered	Filtered	Filtered	Filtered	Filtered
Norm 1	177.1094	177.2508	176.9865	177.2675	178.8301	180.6516
Euclidean norm	206.6165	207.1286	206.3327	206.6682	206.0730	206.1336
Uniform norm	260.9974	261.5478	261.7705	261.8174	262.3801	261.9081
Correlation	0.9618	0.9545	0.9568	0.9565	0.9449	0.9476

	3x3		5x5		7x7	
	Mean filter	**Median filter**	**Mean filter**	**Median filter**	**Mean filter**	**Median filter**
SALT	Filtered	Filtered	Filtered	Filtered	Filtered	Filtered
Norm 1	60.9255	26.2667	63.4635	36.2745	68.6627	48.1098
Euclidean norm	41.1387	3.1515	40.9546	5.3617	40.8882	7.4353
Uniform norm	93.4153	29.6902	89.1564	46.5490	87.6243	56.0471
Correlation	0.8720	0.9787	0.9216	0.9694	0.9283	0.9565

Figure 8: Comparison between the original and the filtered images

2.2 BILATERAL FILTER

Bilateral Filter is an image denoising method [1] that operates in the spatial domain, so it preserves features like edges but cannot preserve low-contrast detail like textures without introducing noise.

The bilateral filter is, then, a smoothing filter for images that replaces the intensity of each pixel with a weighted average of intensity values from nearby pixels. The weights, modeled by a gaussian

distribution, notably depend on differences in color intensity and depth distance, all of which helps preserve sharp edges in the image.

Mathematically the bilateral filter [2] [3] works as follows

$$I^{filtered}(x) = \frac{1}{W_p} \sum_{x_i \in \Omega} I(x_i) f_r(\|I(x_i) - I(x)\|) g_s(\|x_i - x\|)$$

and normalization term, W_p, is defined as

$$W_p = \sum_{x_i \in \Omega} f_r(\|I(x_i) - I(x)\|) g_s(\|x_i - x\|)$$

where

$I^{filtered}(x)$ is the filtered image;

I is the original input image to be filtered;

x are the coordinates of the current pixel to be filtered;

Ω is the window centered in x, so $x_i \in \Omega$ is another pixel;

f_r is the range kernel for smoothing differences in intensities

g_s is the spatial (or domain) kernel for smoothing differences in coordinates

The weight W_p is assigned using the spatial closeness (using the spatial kernel g_s) and the intensity difference (using the range 1 f_r). Consider a pixel located at (i, j) that needs to be denoised in image using its neighbouring pixels and one of its neighbouring pixels is located at (k, l). Then, assuming the range and spatial kernels to be Gaussian kernels, the weight assigned for pixel (k, l) to denoise the pixel (i, j) is given [2] by

$$(i, j, k, l) = \exp\left(\frac{(i-k)^2 + (j-l)^2}{2\sigma_d^2} - \frac{\|I(i, j) - I(k, l)\|^2}{2\sigma_r^2}\right)$$

where are σ_d and σ_r smoothing parameters, and $I(i, j)$ and $I(k, 1)$ are the intensity of pixels (i, j) and (k, l) respectively.

After calculating the weights, normalize them:

$$I_D(i, j) = \frac{\sum_{k,l} I(k, l) w(i, j, k, l)}{\sum_{k,l} w(i, j, k, l)}$$

where I_D is the denoised intensity of pixel (i, j).

MATLAB offers the command [4] to work with this filter:

J = imbilatfilt(I): It receives the image to which you want to apply the filter

J = imbilatfilt(I,degreeOfSmoothing) In addition to the image, it specifies the amount of smoothing.

J = imbilatfilt(I,degreeOfSmoothing,spatialSigma) also specifies the standard deviation of the spatial Gaussian smoothing kernel.

2.2.1 Pseudocode of the Bilateral Filter

Input: Image C

Start
Read C
Select patch Z of C without pointed edges
Trim Z
Find Z variance in Euclidean distance
Assign image smoothing value A (A $>Z$)
Assign parameter A to the bilateral filter
Calculate normalization term G for each neighborhood of pixels Q
Calculate bilateral kernel K of each Q
Calculate noise-free intensity R of each pixel I
Assign value of the bilateral filter with G, K and R of each I of the image
Output: filtered image with value of the bilateral filter in each I
End

2.2.2 Example

We have the original image. We remove it a patch without sharp edges, then we calculate its variance and with this we generate a greater degree of smoothing, we can also extend the spatial extension with a sigma for the variance. We apply the bilateral filter to this image with a degree of smoothing equal to 51.9395 and a spatial sigma equal to 3 and obtain the filtered image.

(a) Original image. (b) Filtered image.

Figure 9: comparison between original image and filtered image

When calculating the correlation coefficient between the two images, we have a value of 0.9994 that tells us that when applying the filter the images almost completely resemble each other, preserving the details and shapes of the original image, but we can see that the bilateral filter improves the background of the image when noise is removed as it looks flatter and more compact.

2.3 GAUSSIAN FILTER

The Gaussian filter is a linear filter that is used in image processing to smooth the details of an image through a convolution process. This process is responsible for transforming two functions into a third one, which will largely represent the magnitude of superposition of the first one on a translated version of the second one, thus eliminating different types of noise that may be in the original image. This Gaussian filter processes images in two different ways with respect to the domain in which you work. For this, it uses the Gaussian function.

$$h(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

as a base and the convolution process is established as $g(x,y) = h(x,y) * f(x,y)$, with a $f(x,y)$ function according to the type of filtering.

2.3.1 Filtering in the frequency domain

In this process the convolution is defined as $G(u,v) = H(u,v) * F(u,v)$[5] where $F(u,v)$ represents the Fourier transformation of the input image and $H(u,v)$ is the frequency attenuating Gaussian filter. The process that is followed is described as

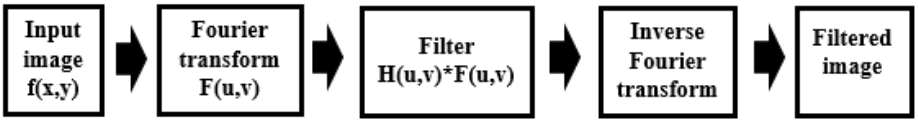


Figure 10: Filtering process with spatial domain

2.3.2 Filtering in the spatial domain

This type of filtering is applied to each of the pixels of the image that is subdivided into nxn sub-matrices, which are known as masks or convolution kernel. For this process, a nucleus $f(x,y)$ is established as the center of the mask and the corresponding convolution with the Gaussian function is performed for each pixel with respect to the nucleus, that is, we look for the approximation in the Gaussian function of the difference of intensity in the image of each pixel and the nucleus. As a result, a sub-matrix is obtained where the nucleus will have a higher intensity value and as the pixels move away from this nucleus they will have a lower value as the image shows.

Figure 11: Filtering process with spatial domain[6]

2.3.3 Pseudocode

To implement this filter, we use the Matlab function $imgaussfilt(A, sigma)$ which works as follows

```
Input: image nxm matrix A
       Standard deviation: sigma
Output: smoothed matix A*

if (A is dispersed) then
    break;
if sigma= ϕ then
    sigma=0.5          #default

apply heuristic method to define domain
if (frequency domain) then
    A*= frequency convolution to A
    A*
if (frequency domain) then
    I ← set of sub-matrices pxq
    While (not every element of I has been analyzed) do
        for (i from 1 to p) do
            for (j from 1 to q) do
                I* ← element of I
                I'= spatial convolution to I*
                A* ← add I' and reorganize image
    A*
```

Figure 12: Gaussian filter pseudocode

2.3.4 Example

To visualize the behavior of the Gaussian filter we have used Matlab Online with a default web image [7]. Here we can see the behavior for different sigma values

(a) Original image.

(b) Filtered image with $\sigma = 0.5$

(a) Filtered image with $\sigma = 1$

(b) Filtered image with $\sigma = 2$

Figure 14: comparison between original image and filtered images

We can see that as the sigma value is greater, the distortion of the image increases and it becomes more blurred. This occurs since the approximation that the Gaussian filter makes for a very large sigma value makes the data much more dispersed and far from its center.

2.3.5 Correlation

We also make the correlation between the original image and the filtered images with the command $corr2(A, B)$ of MATLAB, obtaining the following results.

Correlation between real and filtered image for different sigma	
Sigma	**Correlation**
0.5	99.73%
1	98.01%
2	95.49%

Figure 15: correlation with different σ

It can be seen that, despite the distortion in the image, the correlation is still very high, since it is a method that works uniformly, however, it is notable that the greater the distortion, the lower the correlation.

2.4 WAVELET TRANSFORM METHOD

The Wavelet Transform method is explained below as an option for noise reduction in images. Initially, the image containing all the wavelet transform coefficients is taken, that is, it contains all the information, the next step is to discard those coefficients that do not matter, that is, the parts of the image that contain the noise, to finally with the resulting coefficients obtain the image without noise.

The wavelet transform is a special type of mathematical transform that represents a wave in terms of translated and dilated versions of a finite wave. That is to say, Wavelets are generated from the translation and change of scale of the same wavelet wave called "Mother" as seen in Figure 1, where the blue wave is moved to the right and its scale has increased becoming the red wave.[8]

For an image, the wavelet transform is a transformation of said image in which it is divided into two types of smaller images, the trend (direction or direction of the wave) and fluctuations (variation in intensity, measurement or quality). The trend becomes a lower resolution copy of the original image and the fluctuations store information regarding local changes in the original image. The most significant

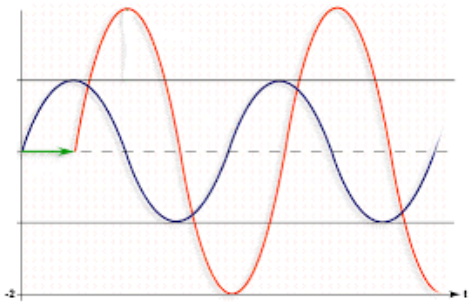


Figure 16: Finite wave, and translated wave and with scale change

trend and fluctuations allow image compression in exchange for discarding irrelevant information and eliminating noise. [9]

2.4.1 Developing

Noise reduction based on the wavelet transform involves three basic steps, which are:

1. Decomposition into wavelet sub bands of the image with noise.
2. Assessment of the wavelet coefficients obtained.
3. Reconstruction of the image from the resulting coefficients using the inverse wavelet transform.

2.4.2 Decomposition into wavelet sub bands of the image with noise

To apply the wavelet transform to images you need to use the two-dimensional (2D) wavelet transform, that is, a dyadic decomposition (2) as shown in Figure 2. The original data matrix B0 (matrix containing the intensity of the pixels of the image) decomposes into 4 sub-bands B1, H1, V1, and D1. Set B1 contains a smoothed image from the original image while H1, V1, and D1 contain the details (reflecting abrupt changes in intensity from one image pixel to an adjacent pixel). Each sub-band is one-quarter of the size of the original image. Then the approximation band B1 can be transformed to obtain B2 and additional details H2, V2 and D2 where the subscripts reflect the level of the transformation. In turn, it is possible to transform B2 and so on.[10]

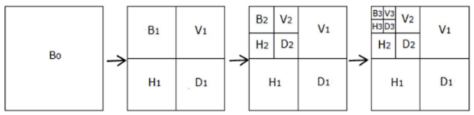


Figure 17: The original image B0 and the first level, second level and third level of the two-dimensional wavelet dyadic (2D) transformation.

This decomposition is presented in Figure 3 with a 256x256 resolution gray-scale image. The image in the upper left corner is the original smoothed image (that is, at a lower resolution of 128x128); it contains the approximation coefficients resulting from applying the wavelet transform. The other three contain the details of the image in vertical, diagonal and horizontal directions. [11]

2.4.3 Assessment of the wavelet coefficients obtained

In this step the relevant coefficients or those with the most important details of the image are determined. This is done through a threshold, a value that allows determining the significance of the coefficients. The coefficient is compared with said threshold, if the magnitude of the coefficient is less

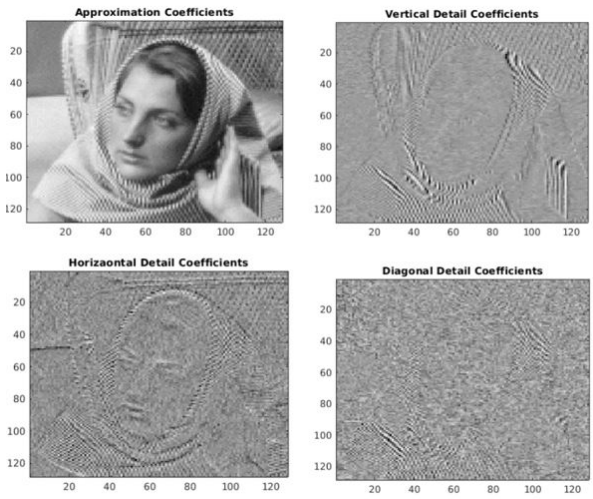


Figure 18: Decomposition of the image by wavelet transform.

than the threshold, this becomes zero, it is declared insignificant (it is eliminated) and if its magnitude is greater than the threshold, it is considered significant (it is not eliminated).

Experimentally it has been shown that the neighboring coefficients of a significant wavelet coefficient are also significant. This is the NeighShrink method that takes into account the magnitude of the coefficient and that of its immediate neighbors to determine if it is significant or not. The experimental results showed that NeighShrink offers better results than the other classic wavelet methods. To make the decision to zero or preserve the coefficient, the magnitude of the coefficient and that of its immediate neighbors are used.

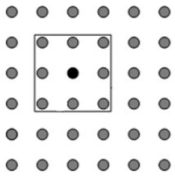


Figure 19: Wavelet coefficient and its neighbors in the same band.

In this way, the threshold distinguishes insignificant coefficients possibly caused by noise from those significant coefficients that are generated by important image structures.

Let $\mathbf{f} = \{f_{ij}\}$ be the matrix that contains the pixels of the original image and $\mathbf{g} = \{g_{ij}\}$ the matrix of the noisy image, we have:

$$g_{ij} = f_{ij} + e_{ij}; i = 1, ..., N; j = 1, ..., M$$

N: number of rows and M: number of columns

Where $\mathbf{e} = \{e_{ij}\}$ represents a white noise matrix with normal distribution $N(0, \sigma_n^2)$. The noise elimination can be formulated as the problem of finding an estimate $\hat{\mathbf{f}} = \{\hat{f}_{ij}\}$ of $\mathbf{f} = \{f_{ij}\}$ from the noise image $\mathbf{g} = \{f_{ij}\}$ that minimizes the error.

Color images can be represented using various arrays (one for each channel, for example three for RGB color images), so without losing generality, you can limit the analysis to gray-scale images.

In the wavelet domain, we get:

$$y_{ij}^{(k)} = x_{ij}^{(k)} + e_{ij}^{(k)}$$

where $y_{ij}^{(k)}$ represents the wavelet coefficients of the complete matrix at the level of decomposition k, $x_{ij}^{(k)}$ the wavelet coefficients of the image without noise and $e_{ij}^{(k)}$ the noise. Thus, in this domain,

the problem is transformed into finding an estimate $\hat{\mathbf{x}} = \{\hat{x}_{ij}\}$ of $\mathbf{x} = \{f_{ij}\}$ from $y_{ij}^{(k)}$ that minimizes the error.

2.4.4 NeighShrink method

In the NeighShrink method for each wavelet coefficient $\mathbf{y} = \{y_{ij}^{(k)}\}$ we need to consider a neighborhood V_{ij} of it. This neighborhood is taken so that it contains the same number of pixels above, below, to the left and to the right of the pixel to consider. This means that the possible sizes of the neighborhood are 3x3, 5x5, 7x7, etc. Figure 3 shows a neighborhood of 3x3 centered on the wavelet coefficient to consider that it is the most used window size and that, as a rule, offers better results. Be

$$S_{ij}^{(k)} = \sqrt{\sum_{(l,m)\in V_{ij}} \{y_{lm}^{(k)}\}^2}$$

the indicator that tells to what extent the area of the image corresponding to the coefficient presents sudden changes in intensity. The estimate $\hat{\mathbf{x}} = \{\hat{x}_{ij}\}$ is obtained from the following expression:

$$\hat{x}_{ij}^{(k)} = \left(1 - \left\{\frac{\lambda}{S_{ij}^{(k)}}\right\}^2\right)_+ y_{ij}^k$$

Where $(\cdot)_+ = max(\cdot, 0)$ and $\lambda = \sigma_n\sqrt{2log(N^2M^2)}$ is the threshold named above. The bound requires prior knowledge of the noise variance σ_n^2. If there is no knowledge of this variance, it can be calculated from the wavelet coefficients $y_{ij}^{(1)}$ of band D1, using the estimator:

$$\hat{\sigma}_n = \frac{Median(|y_{ij}^{(1)}|)}{0.6745}, y_{ij}^{(1)} \in D_1$$

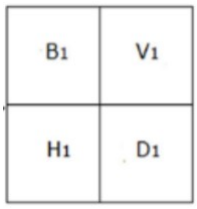


Figure 20: Image of the first level of image decomposition.

With this process the relevant coefficients or those with the most important details of the image are obtained and noise is eliminated.

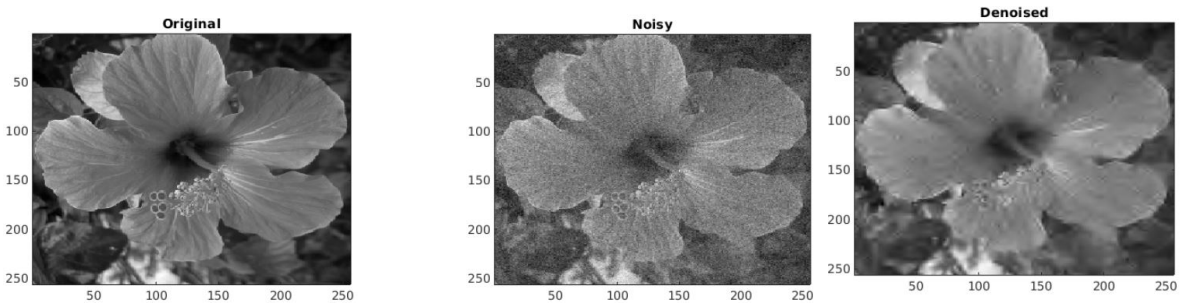
2.4.5 Reconstruction of the image.

Finally, the estimate of the image is obtained through the inverse wavelet transform of the coefficients $\hat{\mathbf{x}} = \{\hat{x}_{ij}^{(k)}\}$. The Inverse Transform takes the found approximation coefficients as input and inverts the decomposition step.

2.4.6 Matlab implementation

Implementation was done in Matlab with the wdenoise2(IM) function which does exactly what was described above.

The figure a) shows a gray scale image, b) shows the same image with noisy and c) shows the denoised image. This example was obtained from Matlab



(a) Original image. (b) Noisy image. (c) Denoisy image.

Figure 21: Comparison original image, noisy and denoisy image.

2.5 Method comparison

Each of the previously seen methods has different characteristics and provide different results, so it is necessary to make a comparison of these to establish which is more appropriate for our research purpose. This comparison was made through the correlation between images, which is based on the maximization of the correlation coefficient determined by analyzing the set of pixels between the two images (original and with noise).

The process consists in choosing different noise intensities for each of the chosen types, which were Gaussian noise and Salt and Pepper. For the first type of noise, intensities of 0.1, 0.2, 0.4 and 0.6 were established, which are equivalent to the variation of the Gaussian noise; This was done for each method obtaining the results presented in the table 24, With the type of noise Salt and Pepper, intensities of 0.05, 0.1, 0.2 and 0.3 were established, equivalent to 5%, 10%, 20% and 30% of noise intensity, respectively.With these results show the images recovered from the noise with the highest correlation values for each case.

The process was done with the figure 22.

Figure 22: Original image

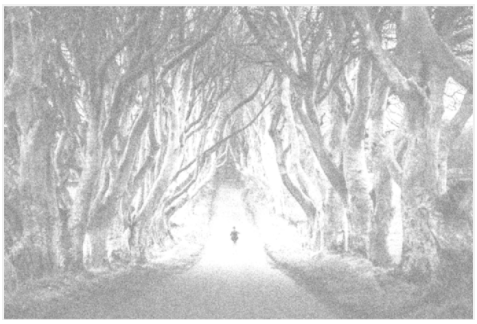
Gaussian noise				
Intensity	0,1	0,2	0,4	0,6
Correlation	0,8761	0,8688	0,8377	0.7442
Salt and Pepper noise				
Intensity	0,05	0,1	0,2	0,3
Correlation	0,7807	0,6461	0,4769	0,3677

Figure 23: image correlation with noise

Image recovery for different noise intensity										
Gaussian noise										
Intensity	0,1					0,2				
Denoise filter	Mean	Median	Gaussian	Bilateral	Wavelet	Mean	Median	Gaussian	Bilateral	Wavelet
Correlation	0,9666	0,9584	0,9715	0,9634	0,9648	0,9640	0,9576	0,9696	0,9592	0,9632
Intensity	0,4					0,6				
Denoise filter	Mean	Median	Gaussian	Bilateral	Wavelet	Mean	Median	Gaussian	Bilateral	Wavelet
Correlation	0,9468	0,9442	0.9551	0,9554	0,9489	0,8885	0,8887	0,9045	0,9039	0,8977
Salt and Pepper noise										
Intensity	0,05					0,1				
Denoise filter	Mean	Median	Gaussian	Bilateral	Wavelet	Mean	Median	Gaussian	Bilateral	Wavelet
Correlation	0,9465	0,9849	0,9575	0,9555	0,7919	0,9381	0,9829	0,9282	0,9142	0,6962
Intensity	0,2					0,3				
Denoise filter	Mean	Median	Gaussian	Bilateral	Wavelet	Mean	Median	Gaussian	Bilateral	Wavelet
Correlation	0,9167	0,9692	0,8619	0,8389	0,6946	0,8872	0,9128	0,7842	0,7826	0,826

Figure 24: Image recovery for different noise intensity

(a) Filtered from gaussian noise with 0.1 intensity



(b) Filtered from gaussian noise with 0.6 intensity

(c) Filtered from salt and pepper noise with 0.05 intensity

(d) Filtered from salt and pepper noise with 0.3 intensity

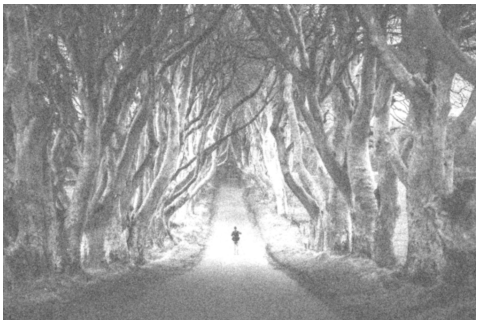
Figure 25: Best filtered image for the given noise

When doing this analysis, we realize that the correlations obtained are very high, but when observing the visual results of the images, these values, in some cases, seem to be unreliable, since the difference with the original image is remarkable. For this reason, we decided to do a statistical analysis of the images by operating them as double matrices and calculating different statistics for a fixed Gaussian noise intensity of 0.25 and a salt and pepper noise intensity of 15%.

Numerical comparison of images - Gaussian noise						
Type of filter	**0,25**	**Mean filter (3x3)**	**Median filter (5x5)**	**Gaussian filter**	**Bilateral filter**	**Wavelet transform**
Image status	Noised	Filtered	Filtered	Filtered	Filtered	Filtered
Norm 1	177,8769	177,1094	177,2675	176,3401	175,9106	177,1412
Euclidean norm	207,0045	206,6165	206,6682	206,9914	207,0005	207,0148
Uniform norm	265,7632	260,9974	261,8174	263,8110	261,8990	262,0394
Correlation	0,8646	0,9618	0,9565	0,9316	0,9656	0.9616

Numerical comparison of images - Salt and Pepper noise						
Type of filter	**0,15**	**Mean filter (7x7)**	**Median filter (3x3)**	**Gaussian filter**	**Bilateral filter**	**Wavelet transform**
Image status	Noised	Filtered	Filtered	Filtered	Filtered	Filtered
Norm 1	74,4157	68,6627	26,2667	66,5954	56,0204	94,4156
Euclidean norm	42,0269	40,8882	3,1515	41,6068	40,4779	42,9052
Uniform norm	99,0118	87,6243	29,6902	95,2990	88,599	137,3251
Correlation	0,5501	0,9283	0,9787	0.7790	0,8969	0,6745

Figure 26: statistical image comparison



(a) best result for Gaussian noise

(b) best result for Salt and Pepper noise

Figure 27: comparison between original image and filtered images

3 Optical Character Recognition (OCR)

"Optical character recognition (OCR) is a process of converting a printed document or scanned page into ASCII characters that a computer can recognize." [12] For the entire develop of this project it is necessary to improve a process where all the characters get recognized once the denoising filter already applied, some techniques Will be used for this porpoise, one of them is PATTERN MATCHING. This method for OCR (optical character recognition) is able to keep a good processing time while maintaining efficiency and versatility. This algorithm has main functional modules which are:

- Image acquisition module
- Pre-processing module
- Feature extraction module
- Pattern generation

Image acquisition module is about obtain a text image from a file. This image was probably scanned(for this algorithm porpoise), so the data may carry some unwanted noise. So, a pre-processor is used to smooth the digitized characters [12]. In the next figure we can find in detail the process of the algorithm.

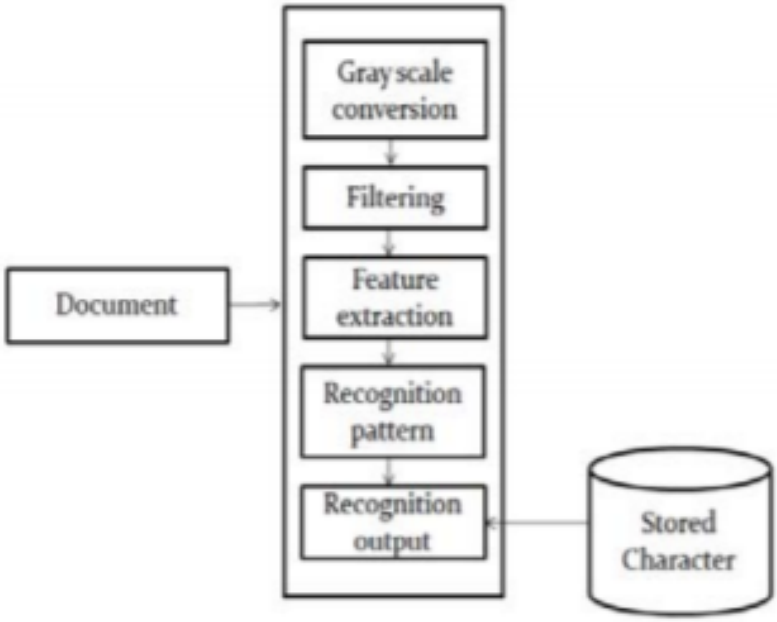


Figure 28: System block diagram

3.1 GrayScale

The first step is to make the image an uniform image in order to analyze every pixel there. To achieve this uniformity in the image it is necessary that the intensity of each pixel is given by general consensus, where a color is converted to an RGB base and from there to a gray scale, which, in this case, is described as follows

$$Y = 0.2126R + 0.7152G + 0.0722B$$

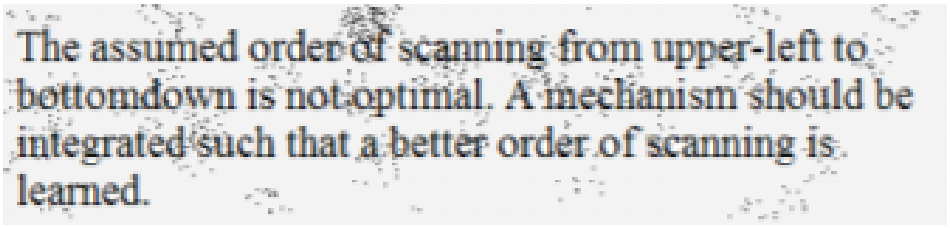


Figure 29: Image without grayscale filter

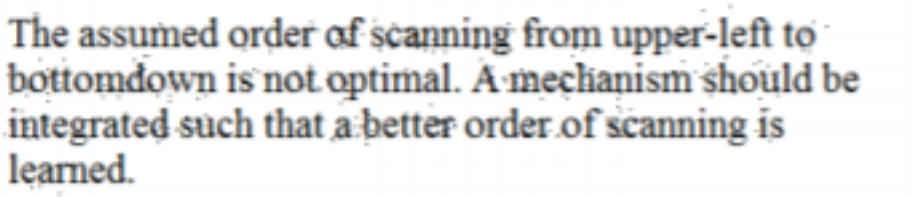


Figure 30: Image with grayscale filter

3.2 Feature Extraction

This is the process of getting information of a specific object in a group of them in order to facilitate classification and test of everyone. The input image might contain some lines of text that needs to be classified into single character for recognition. For this porpoise the following steps are to be applied:

- We analyze the total matrix of the image, scanning from the initial darker pixel, which is about to be named as top of the row.
- Now for the bottom the next blank line is detected. The area between those to items is considered as a row of characters in image.

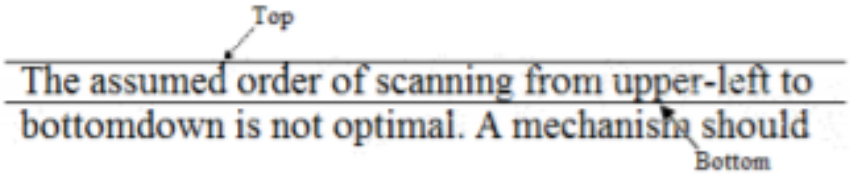


Figure 31: Detecting top and bottom row

- Then, we need to analyze every character in the row. So now this is done scanning the row vertically from top to bottom, the first darker pixel detected is the leftmost pixel in the character of that row.
- If the pixel founded is blank then is the right of the character.

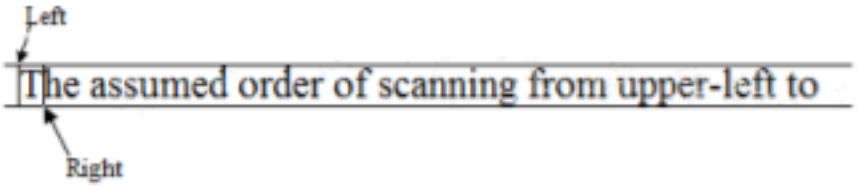


Figure 32: Detecting letter by letter

- Now the image is normalized cutting the edges where the character is white, that's means, cutting the character to the real size of it.

Figure 33: New edge of every letter

- Now we binarize the image we already normalize, where black representing 1 and white representing 0.

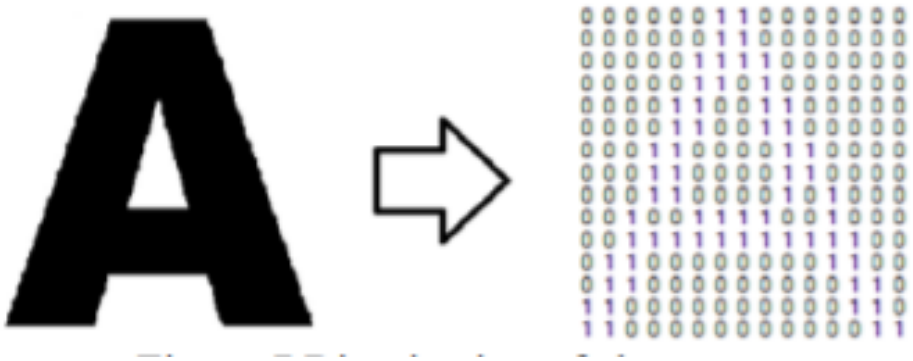


Figure 34: Binarization of the image

3.3 Recognition pattern

The goal in this main module is to compare the image of each trimmed character with that of training data. Due to the fact that each image of characters is made up with numerical pixels, we just need to compare pixel by pixel to find the level of similarity. So that, we calculate the correlation two-dimensional between the unknown data with the data we already have. A higher value means a better match. The formula is:

$$r = \frac{\sum_{m=1}^{M}\sum_{n=1}^{N}\left(A_{mn}-\bar{A}\right)\left(B_{mn}-\bar{B}\right)}{\sqrt{\left(\sum_{m=1}^{M}\sum_{n=1}^{N}\left(A_{mn}-\bar{A}\right)^2\right)\left(\sum_{m=1}^{M}\sum_{n=1}^{N}\left(B_{mn}-\bar{B}\right)^2\right)}}$$

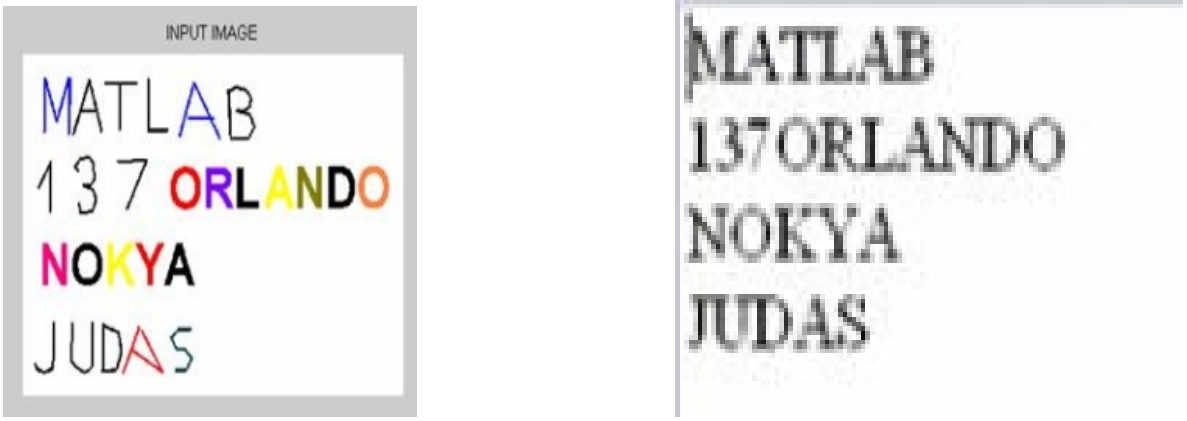
[13]

3.4 Algorithm assumptions

- The algorithm assumes the input image has a clean background, or it has a big contrast with the color of each letter.

- Every character is lined horizontally.
- All the characters have a similar fount and they should be machine printed or similar
- The test data contains characters whose font size is less that 42x42 pixels. (this is the size of the training data).

For example



(a) Proof image(Input) (b) Output

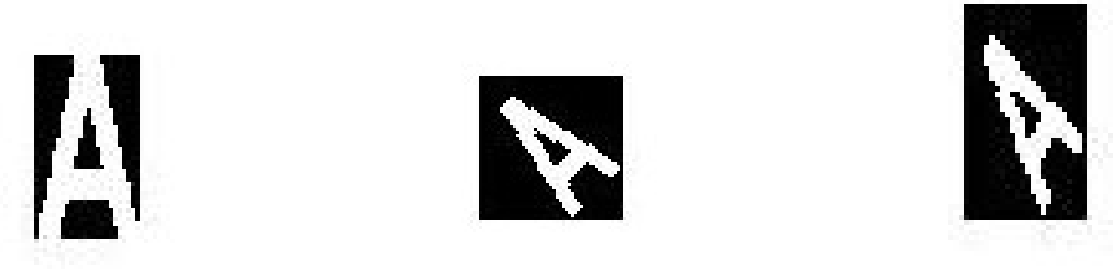
Figure 35: OCR method application in Image (a) and Output in Image (b)

3.5 OCR applied into CAPTCHA recognition

We already now that we have an approach method to recognise characters, but this method need to accomplish a series of ideal assumptions which probably, CAPTCHA images are not able to achieve at all. So, we need to improve this method in order to get a better character recognition. A good way to begin doing this is to increase our training data, considering when characters are rotated, are bigger or smaller, has aditional lines, etc.

3.5.1 Rotation and re-shape

In order to increase our training data, one option is to add the characters in a rotated version. To do this, the main character data base will be taken, then we are going to implement the nearest interpolation method.



(a) Original letter (b) Rotated letter -45° (c) Rotated -45° and re-shaped letter

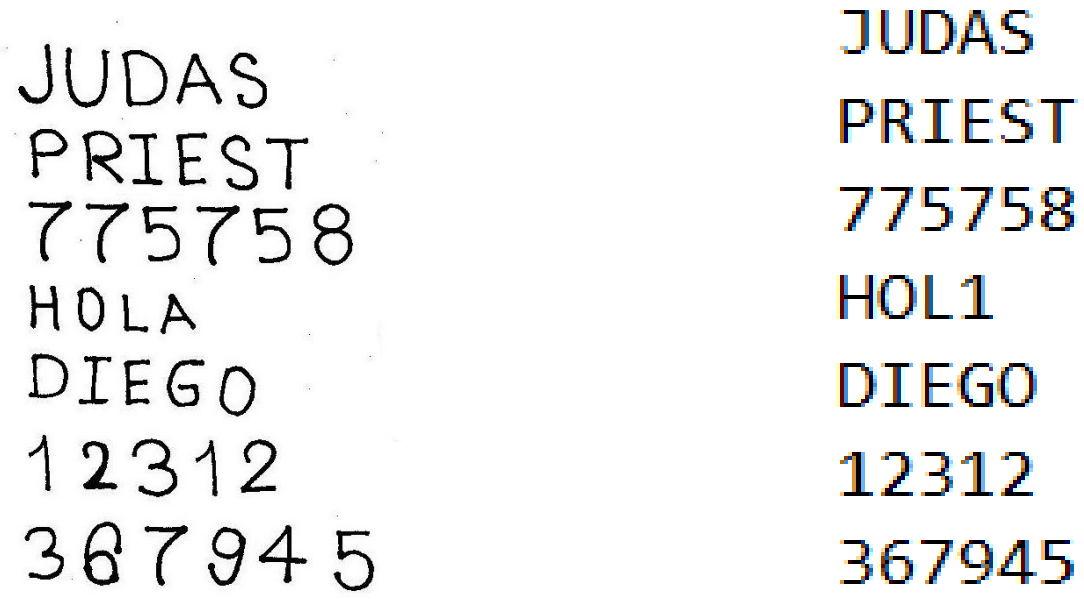
Figure 36: Rotated method process

As it was said before, the main characters to compare with the text need to be all of them in the same size, so it is needed to re-shape the rotated image. Main data base dimensions are 42x24.In the example case we obtain:(figure 32 part c)

In this way, the algorithm will be able to evaluate all this data base fount characters in all possible angles(stressing in angles between -45 and 45 degrees), making it more robust and less sensible to this kind of letter in a text.

3.6 Probability of each option

The algorithm use the Pearson correlation in 2 dimensions to compare the letters, so, this indicator give us a certain number we can evaluate in order to know how feasible and reliable the answer is. So, the algorithm gives a table with the correlation values of each letter. In CAPTCHA applications, this resource could give us a better approximation of how good the algorithm is to certain CAPTCHA formats. For example, given the next image:



(a) Image to see correlation index

(b) Output after OCR algorithm

Figure 37: OCR method application in Image (a) and Output in Image (b)

Caracter	Corr	Caracter	Corr
J	0.6137	O	0.68843
U	0.50781	L	0.70215
D	0.49547	1	0.44776
A	0.35101	D	0.52951
S	0.40383	I	0.74761
P	0.57727	E	0.60028
R	0.46249	G	0.36686
I	0.69058	O	0.60264
E	0.54606	1	0.52917
S	0.43232	2	0.38984
T	0.60193	3	0.46953
7	0.41553	1	0.52917
7	0.41553	2	0.55083
5	0.30548	3	0.48063
7	0.41553	6	0.41651
5	0.30548	7	0.41553
8	0.46939	9	0.42805
H	0.62836	4	0.33436
		5	0.42629

In this table we can see every of the correlation values for the example. It is important to remember that every correlation number here is the maximum value of the correlation analysis with all the character data base.

We can conclude that the algorithm is so good at analyzing the simple character, as "I" or "L", same as "T", "O", and "H". The "worst" one was the number "5". It supposed to be because this number is actually complex to draw.

References

[1] C. Knaus and M. Zwicker. Dual-domain image denoising. In *2013 IEEE International Conference on Image Processing*, pages 440–444, Sep. 2013.

[2] Bilateral filter. In *Wikiwand*.

[3] A. Pardo. Analysis of non local image denoising methods., Ene. 2009.

[4] Bilateral filtering of images with gaussian kernels. In *MathWorks Help Center*.

[5] T. Acharya and A. K. Ray. Image processing., 2005.

[6] G. García. Procesamiento audiovisual. In *Universidad de Murcia*.

[7] 2-d haussian filtering of images. In *MathWorks Help Center*.

[8] Universidad Politécnica de Valencia. Tratamiento de imágenes digitales mediante wavelets.

[9] Václav Hlaváč. Wavelets transformation.

[10] Introducción a la transformada wavelet.

[11] Yanet Cesaire Velázquez Rafael Arturo Trujillo Codorníu and Antonio Cedeño Pozo. Image denoising using interscale and intrascale dependencies between wavelet coefficients. *Revista Cubana de Ciencias Informáticas*, 11(2):1–15, 2017.

[12] Milan Shingote Pratik Ghanwat Faisal Mohammad, Jyoti Anarase. Optical character recognition implementation using pattern matching. In *ISBM School of Technology*.

[13] MATPIC.COM. Reconocimiento de caracteres Ópticos (ocr) usando matlab.