Université de Nice - Sophia Antipolis

A. Y. 2017/2018, Period 1

Long term digital data storage on DNA

Final project Report

Authors:
Contreras Jose Luis
Lo Bianco Riccardo

February 20, 2018

Contents

1 Introduction **1**

1.1 The problem 1

1.2 A bit of history 1

1.3 The present situation 2

1.3.1 Objectives 2

2 Proposed Solution **3**

2.1 Overview 3

2.2 Wavelet Transform 4

2.3 Stack-Run Coding 4

3 Solution Details and Results **6**

3.1 Metrics and testing 6

3.1.1 Test images 6

3.1.2 Metrics: entropy 7

3.2 Wavelet transforms 9

3.2.1 Haar wavelet transform 9

3.2.2 CDF 9/7 wavelet transform 10

3.3 Stack-Run Coding 12

3.4 Scanning 15

3.5 Results 17

3.6 Extension: lossy transformation 18

4 Implementation **20**

4.1 Organization 20

4.2 How to run 21

5 Conclusions **22**

1 Introduction

In this document we will discuss an innovative approach to information storage based on DNA. After a general introduction, we will focus on a specific step of the storage process: the encoding of binary information using the quaternary DNA base.

1.1 The problem

The production and consumption of large quantities of data represents one of the technological trends of the last years. However, data can not simply be produced and used: some kind of storage is necessary in almost all applications. The leading material for data storage devices is silicon, one of the most common minerals on Earth. Silicon would seem like a good solution, since it is abundant and cheap, but many studies agree on saying that, given the current trend of production of new data, the demand of silicon will not be met by the offer in just a few decades. Moreover, this material suffers consistent limitations in terms of data density (quantity of information writable per unit of space) and data retention (number of years before the support starts to degrade).

1.2 A bit of history

The history of DNA encoding starts in 2011, when the researcher Nick Goldman had the intuition that the drastic increase in the speed of production of synthetic DNA could eventually lead to new industrial uses of this technology. In particular, Goldman conceived that DNA could be exploited to store information like a common hard drive disk would do, except for the fact that DNA does not use the conventional binary encoding schema. Before the breakthrough given by this intuition, good results were obtained in terms of speed and costs of production of the DNA strings, but it was not common to think of a usage in terms of mass storage.

A work similar to the one that is nowadays carried on by Goldman's team already started in 1988, when the artist Joe Davis, in collaboration with researchers from Harvard, wrote a binary representation of an ancient symbol in a bacteria's DNA. Similar experiments were conducted in 2011, using a 659 kB book as input signal. However, this last experiment highlighted the importance of error correction in the process: the technology to write DNA strings was still quite young, and the number of errors in the codification of the book was pretty high (22 in the whole document) made it clear that some improvements in the techniques used to write the DNA fragments were still needed.

Nowadays, more advanced approaches are used to the problem of writing and reading DNA strings, but many technological limits are still present.

1.3 The present situation

In 2016, a team from University of Illinois managed to create an approach that allows to access data stored on a DNA string with a random-access approach, empowering the possibility to rewrite encoded data. This result was obtained using mathematical heuristics instead of hard power approach.
This achievement eventually posed DNA on the same level of flash memories for what concerns I/O operations speed, but with clear advantages in terms of data retention and density as well as of power consumption.

However, from a technical point of view, many hurdles are still present. In particular:

- **Scale and speed of DNA fragments' synthesis**, since, at the actual speed of production, the largest storage that could be produced (and its characteristics are not yet released) is the one using the random-access approach cited above. However, this device did not exceed the limit of 0.2 GB and required months for all the DNA to be produced.
- **Cost of DNA synthesis**, since a large part of the expenses sustained along the cited experiments was to be imputed to the cost of DNA synthesis (with values that can go up to 98% of the total budget).

With this background, we can contextualize our work.

1.3.1 Objectives

Our work is focused on the research of an efficient method to write binary data on the quaternary structure of DNA.

With respect to the problems highlighted in the previous paragraph, it is easy to understand the importance that a good compression algorithm can have in this scenario. Moreover, the quaternary structure of DNA and the presence of many biological constraints pose a number of brand new issues for the researchers willing to expand this work, as we will limit our work to the definition of a possible model for the encoding and the subsequent decoding of data written in the two different fashions. Along the discussion, we will provide comparisons of different choices for the implementation of the desired system.

2 Proposed Solution

In this document we describe a solution to write image data in DNA strings based on a combination of traditional image compression techniques and a coding scheme chosen to fit DNA's base-4 nature. This section aims only to briefly present the proposed encoding scheme and focuses mainly on the explanation of the reasons for choosing these particular methods to address this problem. Further descriptions of the chosen methods will be given in later sections.

2.1 Overview

Before diving into the specifics details of this work, we provide a brief overview of the tools and techniques which have been used. In particular, we chose to work with wavelet transform as a preprocessing tool followed by Stack-Run coding. Figure 1 gives a high-level overview of the basic process we followed: we consider a set of input images on which we operate a wavelet transform to have a sparse representation. At this point is where the innovation of our work comes in: we apply the Stack-Run coding algorithm on the resulting representation, as we expect it to provide better compression. After this process, we obtain a 1D vector containing a sequence of 4 different symbols (0, 1, -, + in our case, but the symbols used could have been of any type), suitable to be mapped on the 4 bases of DNA: A, C, T and G. The inverse process works exactly the other way round, taking an input 1D vector and giving the original image as result (notice that, in this case, the process is lossless, so the reconstructed image is exactly the same as the input image).

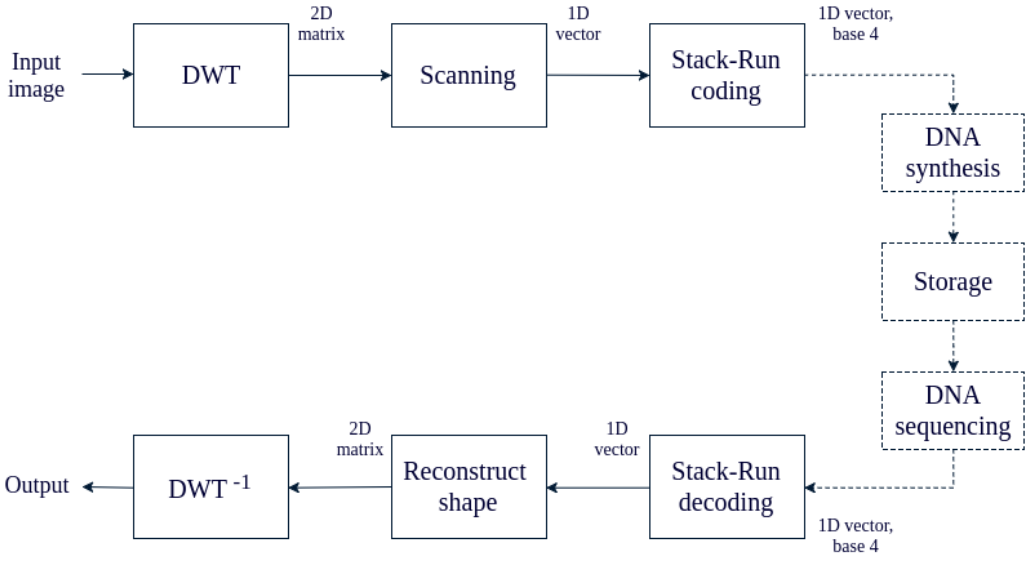

Figure 1: Scheme of the encoding and decoding procedure.

2.2 Wavelet Transform

Wavelet transforms are a well known tool for image compression. The family of wavelet transforms works essentially the same way as the Fourier transform, but, while Fourier transform decomposes the signal into sines and cosines components, i.e. it produces functions localized in Fourier space, the wavelet transform uses functions that are localized in both the real and Fourier (or frequency) space. From a mathematical point of view, we can define a wavelet transform through a function of the form:

$$F(a,b) = \int_{-\infty}^{+\infty} f(x)\psi^*(x,a,b)dx$$

where * is the complex conjugate symbol and function ψ is some function with particular properties. It should be clear that, since ψ can assume many different forms, wavelet transforms are basically a family of methods with a huge set of application fields. In our particular case, we relied on discrete wavelet transform (DWT), which is the term used to indicate any transform for which the wavelets are discretely sampled. We tried two different implementations of DWT: we started with a basic Haar Wavelet, and then we moved to Cohen–Daubechies–Feauveau (9/7) wavelet. In order to avoid using a quantizer, and therefore achieve lossless compression, the proposed transforms are all integer-to-integer.

2.3 Stack-Run Coding

Stack-Run coding [6] is an image coding approach which uses a 4-ary arithmetic coder. Its working principle is to represent significant coefficient values and the lengths of zero runs between them. It performs well when dealing with sparse input data, and therefore it shines when applied to wavelet-transformed images. Moreover, as the produced output is a base-4 vector, it can be directly written into DNA strings without the need of entropy coding algorithms.

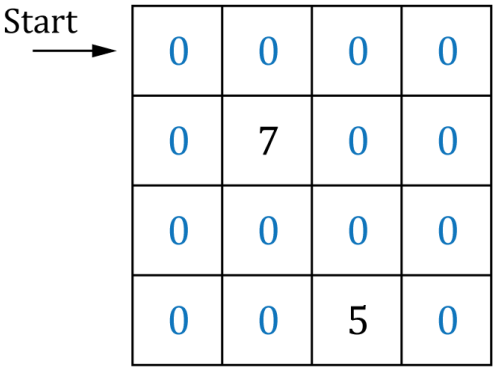

Figure 2: Example matrix representing an image. The arrow points the direction in which values are to be scanned for encoding.

The basic idea behind Stack-Run encoding can be explained taking figure 2 as an example. The image represented by this matrix contains three runs (consecutive zero-valued samples, colored in blue) and two stacks (non-zero samples, colored in black). Thus, the signal would be encoded as a run of length 5, followed by a stack of value 7, another run of length 8, a stack of value 5 and finally a run of length 1. The way to encode these runs and stacks depends on the exact coding scheme chosen. Different implementations of Stack-Run coding exist: we adopted the basic version of the algorithm, extensively described in the following chapter.

3 Solution Details and Results

In this chapter we will provide a complete insight on the techniques introduced previously and discuss the choices we made during their implementation, providing meaningful justifications and explanations.

3.1 Metrics and testing

In this section we introduce the methods we used for interpreting the results presented during the work. Since the results obtained are not treatable with standard techniques (in particular, DNA information storage is quite unconventional, as it does not rely on a binary system, but rather on a quaternary one), we had to slightly redefine well known metrics in order to adapt to the new requirements.

3.1.1 Test images

In order to be able to test methods and compare the results, a set of test images is needed. For this work, we have used a batch of 12 grayscale images of dimensions 512x512. This set contains some of the most commonly used pictures when it comes to image processing testing, such as the classic *lena* or *cameraman*. They have been downloaded from Image Processing Place [1], a website containing different image databases which are useful for this kind of tasks.

(a) Lena.jpg (b) Cameraman.jpg

Figure 3: Two of the images used for testing

3.1.2 Metrics: entropy

In information theory, entropy is a metric used to measure the amount of information produced by a source. It is defined as $H(X) = -\sum_{i=1}^{L} p_i \, log_b(p_i)$, with p_i being the probability of a symbol s_i and L the size of the codebook. The base of the logarithm, b, is typically 2 due to the binary nature of sources. In this work, however, we will be using two different measures, one with $b = 2$ and another with $b = 4$. The unit of measurement is influenced by this aspect: when dealing with the "classical" entropy ($b = 2$), we use bits/pixel, in the quaternary case ($b = 4$) the notation Shannon/pixel is more appropriate. We will choose one metric over the other depending on the stage of the process at which entropy is to be computed, and each of them will have a different interpretation.

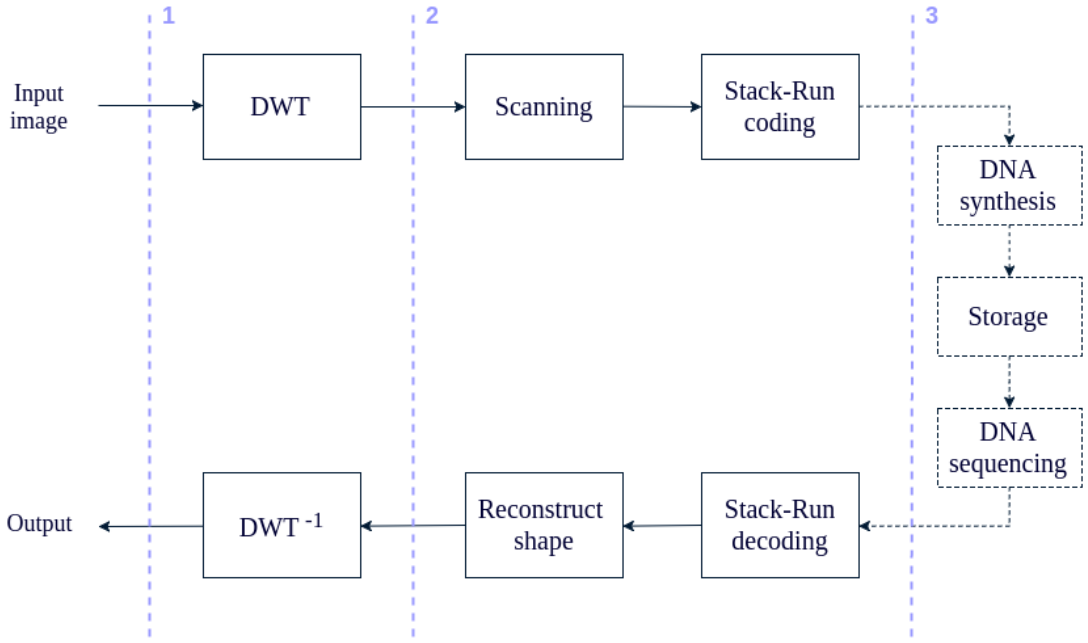

Figure 4: Scheme of the encoding and decoding procedure. Blue markers indicate the stages at which entropy is computed.

Figure 4 illustrates the whole process, with three key points signaled in blue. In the following lines we discuss the different stages at which entropy is calculated and the reasons to keep into account each of these measures:

1. The first measurement (marked as 1 in the figure) is made at the beginning of the chain, and it will serve as a reference to evaluate the efficiency of the process. It follows the classic definition of Shannon entropy, using a base 2 logarithm, and represented in bits/symbol.

2. The second calculation takes place after the quantization step (2). The entropy is computed as a weighted sum of the base 2 entropy of each of the subbands. It is calculated using the same formula as in the previous step, because the aim is to compare it with measurement 1 to evaluate the compression resulting from applying the wavelet transform to the original image.

3. The third measurement takes place after the Stack-Run coding block (3). At this

point, our data is in the form of a vector composed of 4 different symbols, and we define its base-4 entropy as

$$H(Y) = -\sum_{i=1}^{4} p_i \, log_4(p_i) * \frac{len(Y)}{size(X)} [Shannon/pixel]$$

where p_i represent the probabilities of each of the 4 symbols which conform the signal, $len(Y)$ is the length of the encoded vector (in number of symbols) and $size(X)$ refers to the length of the original image (number of pixels). The reason behind this choice is that, at this point, we are interested in minimizing the cost of writing information in real DNA strings, which is an expensive and time consuming operation. As the strings have four bases, the way to optimize compression is to focus on this definition of entropy, rather than on the binary version.

It is important to note that these two different versions of the entropy metric cannot be compared to each other, as they use different bases for the logarithm. Instead, the strategy we will follow is to study the variations in each measure depending on the different techniques and parameters used throughout the process. The first one (used at points 1 and 2 of figure 4) is useful to compare the level of compression obtained with different wavelet transform alternatives. On the other hand, the second metric, which uses a base-4 logarithm, is the most relevant one with regards to writing DNA strings. For small values of entropy in the encoded signal, we require few space to save it, and consequently we save time and money necessary for the synthesis of these strings. The whole objective of this project, then, can be seen as the minimization of the value of this base-4 entropy measure in signals encoded with Stack-Run algorithm.

As we will see in later sections, there is no general method to approach the problem of minimizing entropy in a compression method, at least without losing a part of the information contained in the original signal. For this purpose, we will propose a possible approach based on different scanning of the images in order to process one-dimensional vectors (a necessary transformation for the purpose of DNA writing), then we will resort on quantization to evaluate the opportunity of a lossy approach.

3.2 Wavelet transforms

As already introduced, wavelet transforms are among the most well known tools for image compression. Here we describe the two versions we chose for the implementation of our solution. Again, we must underline the fact that the two methods presented were chosen because both can be implemented in order to produce only integer output results. This is achieved by using a lifting scheme [2], and it allows us to skip the quantization phase, thus obtaining lossless compression.

3.2.1 Haar wavelet transform

Haar wavelet transform is the simplest wavelet transform, often used for testing purposes. With respect to the notation used in the description of wavelet transform, where we wrote the descriptive function as $F(a,b) = \int_{-\infty}^{+\infty} f(x)\psi^*(x,a,b)dx$, the Haar version is characterized by:

$$\psi(t) = \begin{cases} 1 & 0 \leq t < 1/2, \\ -1 & 1/2 \leq t < 1, \\ 0 & else \end{cases}$$

The implementation of the Haar wavelet transform method is rather simple, since it reflects directly the theory behind the mathematical method. What we had to do is simply ensuring that the obtained decomposed image had only coefficients in Z. In particular, given the input array

$$X = [X(1), X(2), ..., X(N)]$$

the lifting scheme takes into account couples of samples $X(i), X(i+1)$ and produces an *approximation coefficient* X_a of the form

$$X_a = \left\lfloor \frac{X(i) + X(i+1)}{2} \right\rfloor$$

and a *detail coefficient* X_d of the form

$$X_d = X(i) - X(i+1)$$

Since we are using the version of the algorithm for 2-D signals, the whole procedure is repeated on rows and columns of the image. Therefore, the Haar wavelet transform is applied first on a per-row basis and subsequently to the columns of the result. Generally, this process is repeated N times, producing as a result the N-levels Haar decomposition of the original image. For a square image of dimensions SxS, the maximum theoretical number of decomposition levels is $N = log_2(S) - 1$. However, it is not always useful to reach this maximum, as very similar results can be obtained with less levels.

Figure 5 plots the average base-4 entropy of the output of transforming the test images, *lena.jpg*, using different levels of decomposition. The entropy decreases significantly at first just to stabilize for values of N greater than 3 or 4. Based on this result, we decided to use 3 levels of Haar decomposition, as the increase in compression after that point does not compensate for the additional complexity.

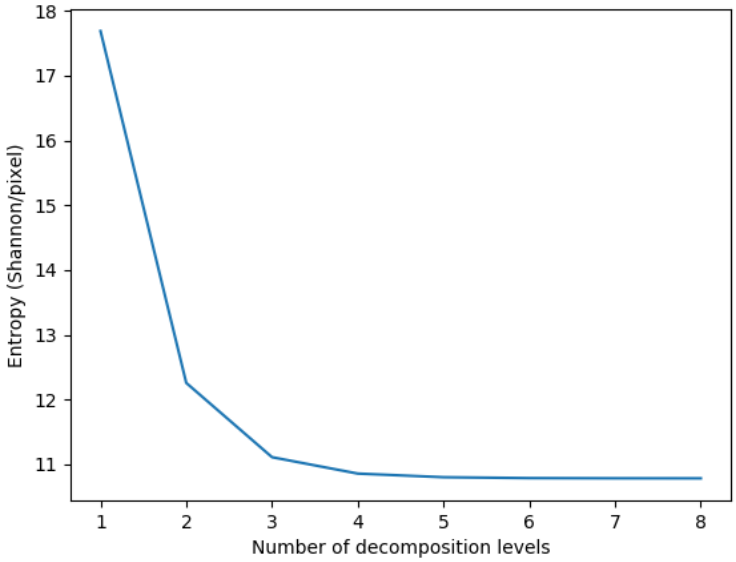

Figure 5: Average entropy of the encoded images versus number of decomposition levels of the Haar transform

3.2.2 CDF 9/7 wavelet transform

Cohen–Daubechies–Feauveau (CDF) wavelet is a famous family of biortogonal wavelets. In particular, two variants became popular since their adoption in the JPEG-2000 standard: CDF 5/3 for lossless compression and CDF 9/7 for lossy compression.

The basic idea of this wavelet family lies in the observation that, for each positive integer value of A, only one polynomial of degree $A-1$ satisfying the following identity exists:

$$\left(1-\frac{X}{2}\right)^A \cdot Q_A(X) + \left(\frac{X}{2}\right)^A \cdot Q_A(X-2) = 1$$

Given the identity below and a value of A, the objective is to factorize the resulting polynomial in the form:

$$Q_A(X) = q_{prim}(X) \cdot q_{dual}(X)$$

in order to obtain a pair of scaling factors of the form:

$$a_{prim}(X) = 2Z^d \left(\frac{1+Z}{2}\right)^A q_{prim}\left(1-\frac{Z+Z^{-1}}{2}\right)$$

$$a_{dual}(X) = 2Z^d \left(\frac{1+Z}{2}\right)^A q_{dual}\left(1-\frac{Z+Z^{-1}}{2}\right)$$

It is interesting to notice that, in the case $A=1$, CDF degenerates in a Haar wavelet. In our task we developed and used the so called 9/7-CDF-wavelet, obtained setting $A=4$. In this case, the polynomial assumes the form:

$$Q_A(X) = 1 + 2X + \frac{5}{2}X^2 + \frac{5}{2}X^3$$

decomposable in:

$$q_{prim}(X) = 1 - cX$$

$$q_{dual}(X) = 1 + (c+2)X + \left(2c + c^2 + \frac{5}{2}\right) X^2$$

The code we used is based on [4] and [3] and it is implemented exclusively for the CDF 9/7 wavelet in lifting scheme implementation: this means that the wavelet transform is represented as a sequence of prediction and update steps. In particular, given the input array

$$X = [X(1), X(2), ...X(2N)]$$

the lifting scheme splits X into two subbands, each of length N:

$$X_o = [X(1), X(3), X(5), ...X(2N-1)]$$

$$X_e = [X(2), X(4), X(6), ...X(2N)]$$

Then, given the filters p and u, the scheme performs the lifting of the subbands to obtain a *prediction step* of the form

$$Xe' = Xe + p * Xo$$

and an *update step* of the form

$$Xo' = Xo + u * Xe$$

where $*$ represents a convolution operator. From the two steps above we obtain the following steps for the 9/7 version:

$$X_e^1(n) = X_e(n) + \alpha(X_o(n+1) + X_o(n))$$

$$X_o^1(n) = X_o(n) + \beta(X_e^1(n) + X_e^1(n^{-1}))$$

$$X_e^2(n) = X_e^1(n) + \gamma(X_o^1(n+1) + X_o^1(n))$$

$$X_o^2(n) = X_o^2(n) + \delta(X_e^2(n+1) + X_e^2(n^{-1}))$$

where the parameters $\alpha, \beta, \gamma, \delta$ take the values:

$$\alpha = -1.586134342$$

$$\beta = -0.05298011854$$

$$\gamma = 0.8829110762$$

$$\delta = 0.4435068522$$

The subbands are then normalized with

$$X_o^3 = k_1 X_o^2$$

$$X_e^3 = k_2 X_e^2$$

where

$$k_1 = 0.81289306611596146$$

$$k_2 = 0.61508705245700002$$

Since we are using the version of the algorithm for 2-D signals, the whole procedure is repeated using X_o^3 as the new X. The inverse transform is done by performing the lifting steps in the reverse order and with $\alpha^{-1}, \beta^{-1}, \gamma^{-1}, \delta^{-1}$ instead of $\alpha, \beta, \gamma, \delta$.

With regards to N, the number of decomposition levels, a similar analysis to the one presented in figure 5 was conducted. The test images were transformed using different levels of decomposition, and the average entropy of the result was calculated. Figure 6 plots the outcome of this experiment. Each additional level of decomposition requires extra calculations, so we opted for a compromise between performance and compression ratio. As the figure shows, entropy is decreasing very quickly with each new decomposition level until $N = 3$, where it stabilizes. The selected number of decomposition levels is therefore $N = 3$.

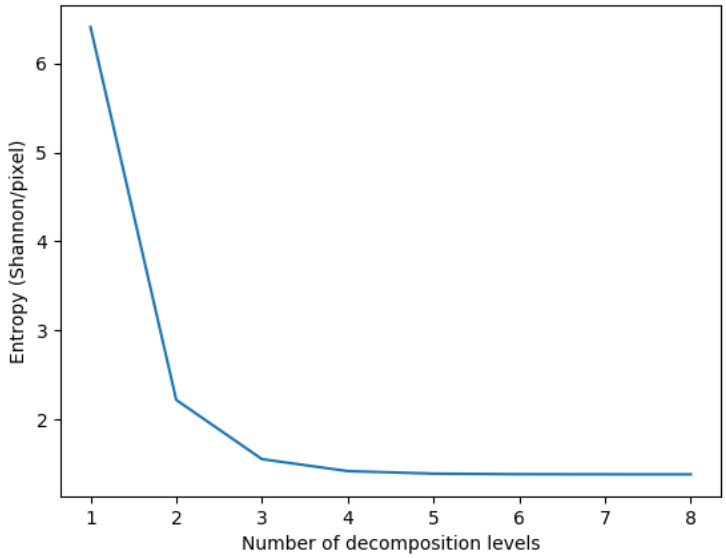

Figure 6: Average base-4 entropy (after SR coding) of the encoded images versus number of decomposition levels of the CDF 9/7 transform.

3.3 Stack-Run Coding

Stack-Run coding is an algorithm studied to maximize the effect on compression ratio of sparsely represented signals. In particular, Stack-Run aims at identifying long sequences of zero valued cells in (n-dimensional) arrays, summarizing the information they carry in fewer space. The basic idea of Stack-Run coding is to clearly differentiate sequences of zeros (runs) from significant values (stacks), using a different policy to encode each of these two classes: runs are treated only considering their length, while stacks are considered in their binary form and treated bit by bit.

We adopted the basic version of the algorithm described in [6]. This implementation fits well the DNA encoding requirements since it uses a four symbols alphabet (namely, "0", "1", "+", "-") that can be easily translated in the AGCT model of DNA. Each symbol has a specific meaning and usage, as shown in table 1.

Symbol	Usage
0	Used for binary 0 in encoding of stacks.
1	Used for binary 1 in encoding of stacks, but not for the most significant bit (MSB)
+	Used to represent the MSB of positive stacks and for binary 1 in representing run lengths
-	Used for the MSB of negative stacks and for binary 0 in representing run lengths

Table 1: Alphabet symbols with relative usage

Given the dictionary above, the algorithm proceeds through the following main steps:

1. As long as the considered samples are zero valued, increment run length
2. When a non zero valued sample appears (i.e. a stack), encode the binary representation of the run length from the least significant bit (LSB) to the most significant bit (MSB)
3. Add 1 to the value of the stack, then encode its binary representation from the LSB to the MSB
4. Loop until the input signal is completely scanned

The flow of the algorithm is intended to encode redundant information in a compact representation, without losing any part of the information. This means that, in the theoretical case where no error is introduced by the machines used for (de)coding and (de)sequencing the DNA strings, the original signal can be perfectly reproduced from the encoded version. The compression effect is obtained thanks to two features of the algorithm:

- Long sequences of zeros are expressed using only the number of consecutive zeros
- The same symbol can be used to encode different fragments of information

The second point is clear if we consider that "+" and "-" are used simultaneously to terminate the sequences of symbols describing runs and to encode the sign, as well as the most significant bit of stacks. Moreover, the algorithm exploits the fact that, since all run lengths are positive numbers, the symbol "+" that would be used to encode the MSB of run lengths can be omitted in most of the cases. The only exception to this rule happens when the run length takes the form $2^k - 1$: in this case, all the symbols used to encode the run length are "+", so it is not possible to eliminate one without introducing an error.

We will now explain all the steps of the algorithm starting from an example input signal. Let's take into account image 7: the image represents graphically the concept of the bit plane and it is easy to distinguish runs (the sequences of zero valued positions) from stacks (the non zero valued positions). In the representation we denote with the (*) arrow the direction of the scanning operation, which impacts on the performance of the algorithm, as discussed in the next chapter.

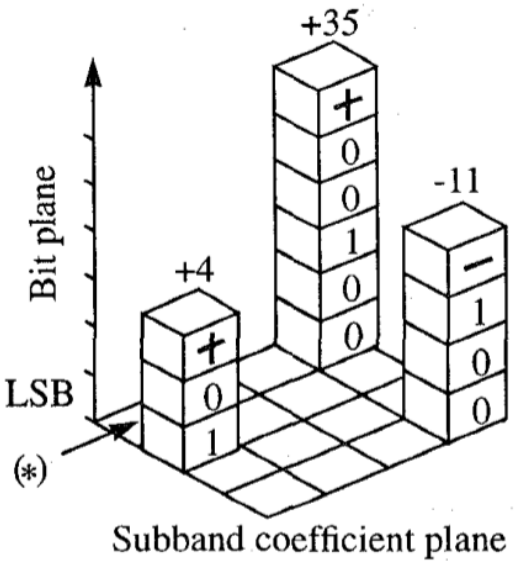

Figure 7: A graphic representation of the bit plane (taken from [6]).

Given this scanning policy, we have a run of length 3 followed by two consecutive stacks, valued respectively +35 and +4, then a run of length 10 and a last stack valued -11. Following the algorithm, we obtain the values depicted in table 2.

Type	**Value/Length**	**Encoded version**
Run	3	++
Stack	+35	00100+
Stack	+4	10+
Run	10	-+-
Stack	-11	001-

Table 2: The results of the application of the Stack-Run algorithm

We can notice that the first run is of the form $2^k - 1$ ($3 = 2^2 - 1$), so it has binary encoding 11, translated into ++ by the Stack-Run dictionary. The last + can not be elided in this case. On the contrary, a run with length 10 (1010 in binary notation) would be encoded as $- + -+$, but the algorithm rules allow to erase the last +. The absence of runs between two stacks does not result in the introduction of unnecessary sequences. Concerning the stacks, we must underline the fact that the elision is never possible since the value could be either positive or negative. The output of the algorithm is always a one-dimensional vector, so the reconstructed version must take into account the position of the different samples in the original image. We discuss this issue in the following section.

3.4 Scanning

Due to the nature of the Stack Run coding scheme described above, results may vary depending on how the image is scanned through before being encoded. In order to make the most out of this algorithm, it is within our interest to maximize the number of consecutive stacks. This way, the encoded image will contain less symbols, as one long run takes up less symbols to encode than two shorter ones.

In our approach, each subband of the transformed image is scanned independently. Following the reasoning explained above, it was decided to use different scanning directions depending on the characteristics of each subband. Due to the nature of wavelet transforms, we expected each of the subbands to have slightly different features: some of them reflect vertical edges, while others highlight horizontal ones. To take advantage of this fact, we propose a mixed scheme: bands where the vertical components are stronger are to be scanned vertically, and the rest will use horizontal scanning. Figure 8 provides a visual example.

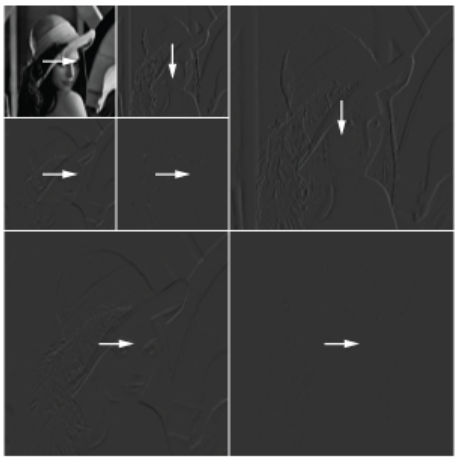

Figure 8: Scanning directions for each subband on a sample level 2 Haar decomposition of Lena.jpg.

To test this technique, we encoded each of the test images using three different scanning approaches:

1. The mixed approach described above (see also figure 8), using different directions depending on the characteristics of each subband.
2. Purely horizontal scanning on all subbands
3. Vertical scanning on all subbands

The results, resumed in table 3, do not show any appreciable improvement in the algorithm's performances. The entropy values differ slightly from one policy to another, but no pattern suggests that one approach is superior to the rest. However, we must highlight that the adoption of a policy of scanning by subbands brings immediate benefits: in particular, the dimension of the input signals decreases significantly (at most, we will consider only one quarter of the original image, given the properties of wavelet transforms) and

this aspect results in a massive decrease of the size of encoded version, since it is possible to transmit each subband separately. Given the biological characteristics of DNA strings, having short sequences is a fundamental property for an efficient storage system. With this aspect in mind, we decided to encode each subband singularly following an horizontal scan.

Image	**Mixed**	**Horizontal**	**Vertical**
lake.jpg	2.031	2.030	2.031
woman_blonde.jpg	1.947	1.947	1.947
woman_darkhair.jpg	1.787	1.786	1.786
cameraman.jpg	1.892	1.890	1.891
lena.jpg	1.886	1.886	1.885
jetplane.jpg	1.906	1.905	1.906
house.jpg	1.828	1.823	1.832
peppers_gray.jpg	1.929	1.929	1.929
livingroom.jpg	1.964	1.964	1.964
pirate.jpg	1.971	1.971	1.971
mandril_gray.jpg	2.064	2.064	2.064
walkbridge.jpg	2.101	2.101	2.101

Table 3: Comparison of the base-4 entropy (Shannon/pixel) of the test images, once encoded using the different scanning approaches.

3.5 Results

The process described above was applied to the images in the test set, and the results are summarized below. The goal here was to compare the performance of the different methods explained in sections 3.2.1 (Haar wavelet) and 3.2.2 (CDF 9/7 transform). As discussed previously, both transforms are applied with 3 levels of decomposition and followed by Stack-Run coding. Finally, the transformed images were scanned through horizontally, since no particular benefit arose from other scanning policies.

		Wv transform (bits/px)		**SR encoded (Shannon/px)**	
Image	**Original entropy**	**Haar**	**CDF 9/7**	**Haar**	**CDF 9/7**
lake.jpg	7.49	5.44	2.91	3.01	1.55
woman_blonde.jpg	7.16	5.15	2.53	2.88	1.36
woman_darkhair.jpg	7.27	3.77	0.87	2.13	0.51
cameraman.jpg	7.05	4.08	1.11	2.25	0.61
lena.jpg	7.45	4.67	1.78	2.61	0.95
jetplane.jpg	6.72	4.50	1.69	2.49	0.91
house.jpg	5.77	3.36	0.56	1.74	0.38
peppers_gray.jpg	7.58	5.09	2.57	2.88	1.42
livingroom.jpg	7.44	5.24	2.66	2.92	1.43
pirate.jpg	7.31	5.14	2.47	2.84	1.30
mandril_gray.jpg	7.29	5.79	2.24	3.29	1.29
walkbridge.jpg	7.71	5.99	3.71	3.37	2.01

Table 4: Comparison of the base-4 entropy (Shannon/pixel) before and after encoding with for each of the test images.

Table 4 shows the decrease of entropy from the original image and the two versions of wavelet decomposition (both measures are expressed in bits/pixel) and the final result expressed in Shannon/pixel (refer to section 3.1.2 for details on both metrics). Once again, it is important to underline that the result shown in the last two columns is not directly comparable with the other numbers in the table, since the measure units are different.

With regards to the entropy of the wavelet decompositions, it is easy to notice that the CDF 9/7 wavelet transform greatly outperforms the Haar wavelet implementation. The Haar approach offers compression rates ranging from roughly 1.5 to 2, while the CDF 9/7 transform provides compression rates up to 10 in some cases. This is an expected result since CDF 9/7 is well known as one of the most powerful wavelet implementations, while Haar wavelet is commonly used for experimental scopes, but rarely for deployment.

The base-4 entropy of the encoded strings also benefits from the usage of CDF 9/7. In this case, working with this transform instead of Haar results in an improvement by a factor of 2 in the worst cases, up to a factor of 4 for some images.

3.6 Extension: lossy transformation

The schema presented until this point (figure 4) aims only at lossless compression. In order to do so, we have been using only integer to integer wavelet transforms, thus avoiding the need to use a quantizer and the subsequent quantization error. However, quantizers bring consistent improvements in terms of compression rate, and this section aims to present an alternative approach which benefits from them.

Figure 9 illustrates the scheme of the new approach. The integer to integer wavelet transform which was being used previously has been substituted by a biorthogonal 2.2 wavelet [5], which produces decimal coefficients. The output of this transform is then quantized with a uniform scalar quantizer, using different quantization steps for each of the subbands. In particular, during this experiment, the quantization step has been kept constant for all subbands corresponding to the same decomposition level of the wavelet transform, using smaller steps for those subbands associated with the last levels (the smaller ones). In order to have a fully optimized procedure, the quantization step should be different for each subband and decided basing on the results of a dual programming optimization routine, but we decided not to implement this part of the procedure because it is not necessary to obtain meaningful results, as we will see. In particular, we applied a quantization step of $\delta = \frac{256}{2^{n+3}}$, where n is the dec level (smallest subbands have $n = N - 1$, where N is the total number of levels of decomposition in the image, while the biggest ones have $n = 1$). For future use, the predefined setting can easily be modified.

Apart from the introduction of the quantization module, the process remains the same as before: the subbands are scanned to produce a 1D vector, which is then fed to the Stack-Run encoder. As explained in section 3.3, the output of the Stack-Run coding is a base-4 vector which can be written in DNA strings.

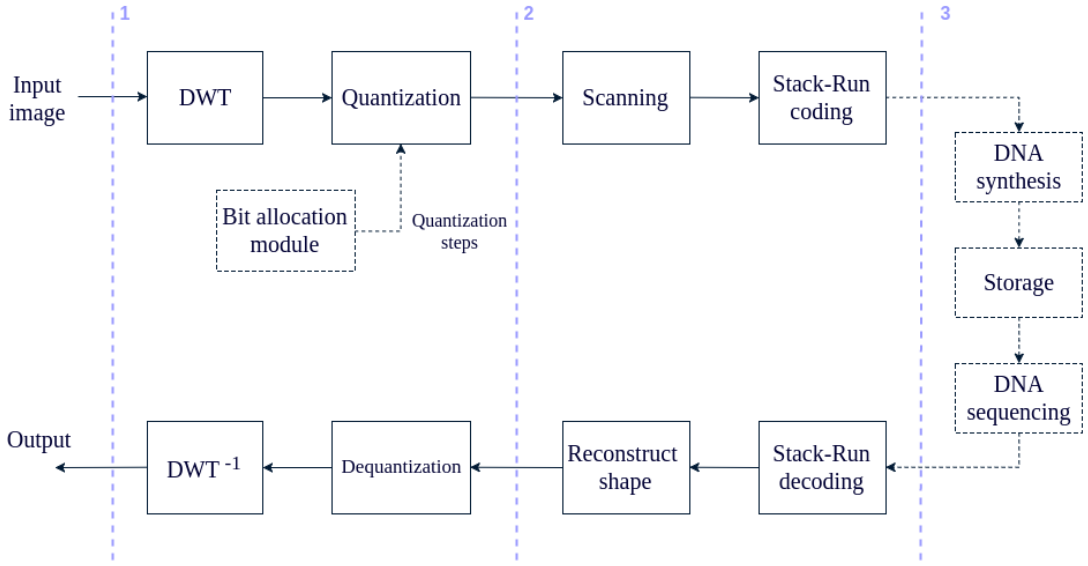

Figure 9: Scheme of the encoding and decoding procedure, including the required modules for the quantization extension.

In order to measure the efficiency of this approach, the metric we will use is entropy, following the same approach as in the previous experiments (details in section 3.1.2).

Again, we focus on the entropy in three key points of the processing chain, signaled in blue in figure 9:

1. As in the previous case, the purpose of first measurement (marked as 1) is to serve as reference to evaluate the compression capabilities of the two following blocks. It follows the classic definition of Shannon entropy, using a base 2 logarithm, and represented in bits/symbol.

2. The second entropy calculation is now done after the quantization step (2). The contribution of each of the subbands to the total entropy is computed and added up to determine the entropy of the transformed, quantized image. The goal here is to test the compression ratio obtained by the first two modules (wavelet transform and quantization). Thus, the entropy is computed in the same way as in the first step.

3. The third measurement is equivalent to the one in the previous experiment (explained in section 3.1.2), and therefore it will allow us to verify the impact this approach has on the final result (remind that this metric is related to the length of the DNA strings required to write our data, and thus it is the one we will base our decisions on).

	Entropy			
Image	**Original**	**Quantized (bits/sym)**	**Encoded (Shannon/px)**	**MSE**
lake.jpg	7.49	1.70	1.02	14.20
woman_blonde.jpg	7.16	1.54	0.92	11.82
woman_darkhair.jpg	7.27	0.73	0.46	3.46
cameraman.jpg	7.05	1.00	0.58	4.11
lena.jpg	7.45	1.18	0.71	7.66
jetplane.jpg	6.72	1.19	0.71	6.15
house.jpg	5.77	0.68	0.43	1.99
peppers_gray.jpg	7.58	1.43	0.91	14.80
livingroom.jpg	7.44	1.60	0.95	11.22
pirate.jpg	7.31	1.54	0.92	10.39
mandril_gray.jpg	7.29	1.64	0.99	8.79
walkbridge.jpg	7.71	2.19	1.34	16.87

Table 5: Results of the lossy compression process. Entropy values are evaluated at the points shown in figure 9. MSE is calculated between the quantized wavelet transformed image and the original image.

Table 5 shows the results of the complete process, including quantization on the wavelet transformed image. By comparing these results with those depicted in table 4 we can notice that the impact of quantization on the process is extremely significant. The entropy measured in Shannon/pixel on the Stack-Run encoded signal is lower than its lossless counterpart in all cases, at the expenses of a non critical MSE (measured between the original image and the image encoded with wavelet transform). The resulting decoded images present some artifacts, but their impact is so low that it is difficult to spot them without an accurate analysis.

4 Implementation

In this section we will provide a complete handbook for the use of the Python code we created during the development of the project. This guide is intended for those willing to extend and integrate the work presented until now.

4.1 Organization

The code for the project is contained in a GitHub repository at the address `https://github.com/lobiaminor/dnaEncoding`. The content of this repository is:

1. A *main.py* file containing the executable code handling wavelet transformation and Stack-Run (encoding and decoding) on target input images. This program prints in output the results of the compression, in terms of entropy, and it is capable of writing the strings resulting from Stack-Run coding to .txt files for easy reuse.
2. An initialization file *settings.ini* containing the parameters to be used in the main program, in particular:
   - *imgdir*: the path to the location of the input images.
   - *extension*: the extension of the images to be processed (example values are *jpg*, *png*, *dat*).
   - n: the number of decomposition levels to be used in the wavelet transform. As discussed along the document, a coefficient $n = 3$ should be adapt to most of the applications, but we decided to make it easy to adapt the parameter to different requirements.
   - *mode*: the type of compression we want to obtain. Possible values are *lossless* (default), meaning that the wavelet transform is directly followed by Stack-Run encoding, thus leading to a lossless compression, and *lossy*, meaning that a quantization is performed on the wavelet transformed image and the result is used as input for the Stack-Run encoder, as previously discussed.
   - *saveresults*: option to disable the creation of files containing the results of compression (the strings resulting from the Stack-Run encoding). Default is 1.
   - *outputfolder*: the path to the folder where the results will be saved (if *saveresults*=1).
   - *suffix*: the suffix to append to the name of the results to distinguish them from the original input images (default is *encoded.txt*).
3. An *img* folder containing the set of test images.
4. Two files, *haar.py* and *cdf97.py*, files containing respectively the code for wavelet encoding/decoding using Haar or CDF 9/7 (integer to integer).
5. A *stackrun.py* file containing the code for encoding decoding of a given signal using the Stack-Run algorithm.

6. A *miscellaneous* folder containing various scripts used for the evaluation of the strategies we tested. We will not enter in the details of these scripts here, we made sure to comment them properly for future reuse.

7. A *README.md* document containing a brief description of the project scopes and a resume of the configuration parameters.

8. A *dnacodingdocs* folder containing various documents regardig the arguments treated treated (and this document).

4.2 How to run

In order to run the code for the projec, a Python 3.6 installation is necessary and a set of dependencies must be fullfilled. To run the code, it is necessary to download the full GitHub repository from the address specified above, move into it and use the command

```
python main.py
```

With a basic Python installation, the code will probably not work because of the absence of some dependencies. If this happens, please refer to the nature of the missing dependency and install the corresponding package using the command:

```
pip install name_missing_package
```

Once the basic version of the code runs properly, it is possible to change the basic settings contained in the file *settings.ini* to verify the effects on the output and adapt code to further improvements.

Note: the current implementation of the integer to integer CDF 9/7 is only capable of handling square images due to a lack of software engineering in the code we used as a base for ours, which makes it really hard to adapt it to non squared input images. This kinds of problems always require a lot of time to be solved properly, we will make sure to do it in a future improvement.

5 Conclusions

Along this work we analyzed a possible approach to DNA coding based on a set of well known compression techniques and on Stack-Run, an algorithm for the encoding of quaternary signals.

The encoding method we presented represents the first step of a larger task. In particular, what we developed can be seen more as a theoretical approach to quaternary coding rather than a complete implementation of a DNA coding algorithm. In fact, the translation of the four symbols produced by Stack-Run coding (1, 0, +, -) into the four nucleotides constituting DNA strings (A, C, T, G) can not be performed with a simple one to one relation due to the nature of the chemical processes used to synthesize and sequence DNA. Current procedures are more significantly more prone to errors when dealing with some particular cases (e.g. when the percentage of GC is high it is more likely to produce an error during the chemical synthesis). Therefore, in order to obtain successful applications of the algorithm, a further study on possible translation schemes from Stack-Run coding to viable nucleotides' sequences must be performed.

To evaluate the goodness of the resulting signals, we defined a modified version of the classical entropy for signals encoded on four symbols. This point must be kept into great consideration because of its novelty: it is rare to find compression-related works developed for non binary settings, and thus the use of modified versions of the entropy measure is one of the crucial points of our project. The base-4 entropy is not comparable with the classical base-2 entropy, and thus it is not adaptable for the evaluation of the level of compression obtained by Stack-Run coding with respect to the previous step in the compression algorithm. This kind of comparison would not make sense, since the means of storage used in the two cases differ in their intrinsic characteristics. The aim of defining a base-4 entropy is to give a base for the comparison between the results obtained applying Stack-Run coding and future algorithms designed for encoding signals in DNA. In this sense, it is important to underline one last time the fact that the results we obtained can not be considered in absolute sense, on the opposite a term of comparison is necessary and still missing.

We discussed the possibilities offered by a lossy compression method based on scalar quantization of the wavelet transformed image. We decided to implement this section to evaluate the magnitude of the impact on the length of encoded signals with this approach after observing that, given the state of art of technologies for DNA sequencing, the size of the signals encoded with the lossless approach is still too big, leading to prohibitive requirements in terms of time and resources. The lossy approach provides an easy and powerful solution to this issue, but the trade-off between compression ratio and magnitude of the introduced error is a critical aspect for the final scopes of DNA encoding: if we think of DNA as a promising support for long term memories used for backup of rarely accessed information, we must keep into account the fact that an incorrect encoding could lead to irreparable losses of information. Moreover, the machinery used for reading DNA strings is still heavily prone to errors (whose incidence depends on the model used) due to biochemical constraints, thus the impact of a propagation of errors in a real world scenario is a possibility that must be taken into account. On the other hand, by looking at the results in terms of entropy values, it is clear that the advantages are significant and the

mean square error values we measured are quite low, so we think that this possibility is to be taken into account in this first phase of the study.

As already mentioned, the results obtained through Stack-Run are still too long to be considered for extensive use in DNA coding. This leaves much space to further developments of the work presented in this document, in particular in terms of optimization of the signal given as an input to the Stack-Run algorithm. With this objective, we presented a possible improvement based on different scanning policies of the wavelet transformed images, but the results did not present appreciable improvements. This fact does not mean that more sophisticated methods could not provide important reduction in the size of the final output of the Stack-Run algorithm nor that a smart scanning policy could not improve the results of a different quaternary coding algorithm. For the moment, a lossless solution to the problem of the output length would be to fully exploit the characteristics of wavelets by encoding each subband in a different DNA string and use the set of produced strings for the reconstruction.

Given all these motivations, we conclude by saying that there is room for a great variety of improvements to the presented work, in particular towards an exhaustive search of all possible methods to be used in the process as we designed it. Haar and CDF 9/7 are only two of the many existing wavelet implementations, different variants of the Stack-Run algorithm were presented and, even regarding the lossy approach, a good improvement can be obtained by optimizing the size of the quantization steps. On the other hand, we believe that the present work can constitute a foundation for future researches also thanks to its methodical approach to the presented problem.

References

[1] Image processing place: Image databases. http://www.imageprocessingplace.com. Accessed: 2017-12-06.

[2] AR Calderbank, Ingrid Daubechies, Wim Sweldens, and Boon-Lock Yeo. Wavelet transforms that map integers to integers. *Applied and computational harmonic analysis*, 5(3):332–369, 1998.

[3] Pascal Getreuer. Wavelet cdf 9/7 implementation. http://www.getreuer.info/home/waveletcdf97. Accessed: 2017-12-18.

[4] Victor Olhovsky. 2d cdf 9/7 wavelet transform in python. http://www.olhovsky.com/2009/03/2d-cdf-97-wavelet-transform-in-python/. Accessed: 2017-12-18.

[5] Pywavelets. Biorthogonal 2.2 wavelet properties, filters and functions. http://wavelets.pybytes.com/wavelet/bior2.2/. Accessed: 2018-02-16.

[6] Min-Jen Tsai, JD Villasensor, and Feng Chen. Stack-run coding for low bit rate image communication. In *Image Processing, 1996. Proceedings., International Conference on*, volume 1, pages 681–684. IEEE, 1996.