Master Project

Point cloud compression for DNA based storage

Author:
Romain Graux

Supervisors:
Prof. Dr. Touradj Ebrahimi
Davi Lazzarotto

Chapter 1

Introduction

Each year, we produce more data than the previous one. All those datas are stored in multiple datacenters spread all over the world. Those datacenters require a lot of energy to maintain bits of information in spinning disks, tapes, capacities, transistors, ...

The Independent reported in 2016 that data centers will consume three times as much energy as they are currently using over the course of the next decade. [1]

It becomes naturally important to find more eco-friendly ways to store data.

Chapter 2

Problem defintion

In this work, we will try to build an end-to-end point cloud compression model for quatarnary based entropy coding.

We will first discover the state of the art of compression models for point cloud that leads to the best bitrates for various point clouds and then adapt it to output **A**, **C**, **G** and **T** symbols instead of the classical 0 and 1 binary base.

Then we will have to study the state of the art of DNA based storage and how we can store our long sequence of **A**, **C**, **G** and **T** in the most efficient way to minimise the cost of storing those datas while trying to recover the compressed point cloud in the most XXXX way. For this last part, we will have to study the influence of each parameters to store the DNA strand as small chunks in a solution. At the end, we will have to propose the best parameters to store a DNA strand for a certain period of time.

And finally, we will have to provide the reconstruct model to go from the compressed version to the point cloud as closed as possible from the original one.

To continue

Chapter 3

State of the art

3.1 DNA storage

As we produce more data every year, we need to find a way to store it efficiently. Currently, we store our data in big data center consuming a lot of energy to keep these informations in electronic devices, **Find the consumption of data centers**

It would be a good idea to find a way to store our data in a more efficient and ecological way. For storing information, hard drives don't hold a candle to DNA. Our genetic code packs billions of gigabytes into a single gram. A mere milligram of the molecule could encode the complete text of every book in the Library of Congress and have plenty of room to spare. [2]

But it can not be applied to all data types, for example, it is not possible yet to replace an USB stick by a DNA based USB stick and expecting the same experience. The information retrieval latency and high cost of the DNA sequencer and other instruments "currently makes this impractical for general use," says Daniel Gibson, a synthetic biologist at the J. Craig Venter Institute in Rockville, Maryland, "but the field is moving fast and the technology will soon be cheaper, faster, and smaller." Gibson led the team that created the first completely synthetic genome, which included a "watermark" of extra data encoded into the DNA. [2]

This does not mean that there are no applications for DNA based storage. DNA based storage can be used for long term media preservation archives (so called cold media storage) which are infrequently accessed and thus do not need low information retrieval latency.

3.1.1 DNA

Desoxyribonucleic acid (DNA) is a chemical molecule composed of two polynucleotide chains that coil around each other to forn a double helix carrying genetic information. DNA is the primary information carrier in living organisms, it is the building block of all life carrying instructions for the development, functionning, growth and reproduction of all known organisms and many viruses.

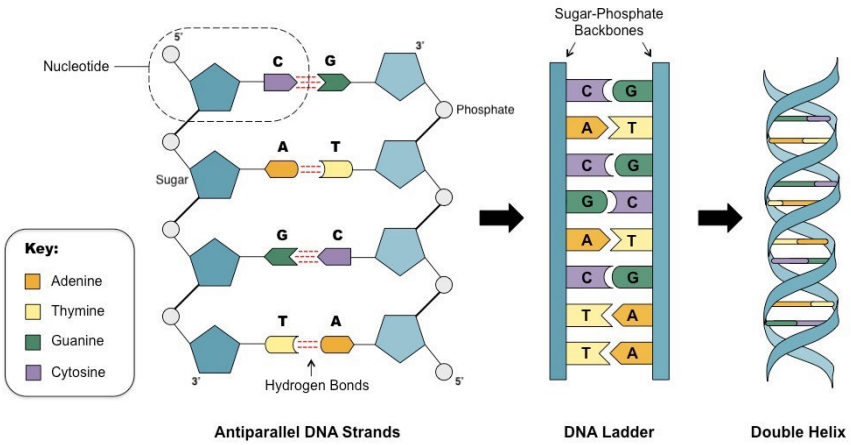

Figure 3.1: DNA structure

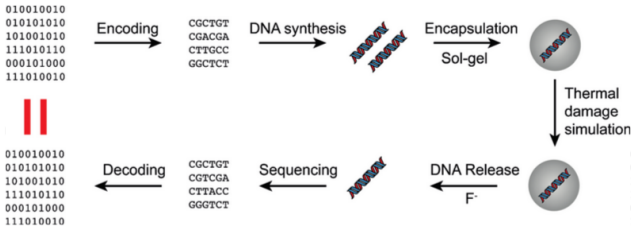

Figure 3.2: DNA storage pipeline

The two chains are known as polynucleotides as they are composed of monomeric units called nucleotides as described on Figure 3.1. Each nucleotide is composed of one of the four nitrogen bases (adenine [A], cytosine [C], guanine [G] and thymine [T]), a sugar called deoxyribose and a phosphate group. The nucleotides are linked to one another in a chain by a covalent bond between the sugar of one nucleotide and the phosphate group of the next, resulting in a alternating sugar-phosphate backbone. The nucleotides of the two separate chains are joined together by an hydrogen bond between their nitrogen bases according to base pairing rules (A-T, G-C) resulting in a double helix. The nitrogen bases are divided into two groups, the pyrimidine bases (C and T) and the purine bases (A and G).

Both strands of DNA store the same information but in complementary pairs. The two strands of DNA run in opposite directions and are thus antiparallel. The direction is determined by the 5' to 3' or 3' to 5' direction.

3.1.2 Usage

DNA is a powerfull tool for storing information for biological systems so it can also find its way in the DNA storage. DNA storage can be used for applications that require:

- **High information density**: In 2017, scientists at Columbia University and the New York Genome Center published a method which allows perfect retrieval of information from a density of 215 petabytes per gram of DNA [3];
- **Data longevity**: Information stored in DNA can last for decades or centuries compared to conventional media which are replaced every 3-5 years [4];
- **Low energy consumption**: DNA does not use energy to store information, energy is only used to help the degradation of the DNA;
- **Ease of replication**: Thanks to the Polymeras Chain Reaction, DNA can be replicated easily [5].

3.1.3 End-to-end DNA storage

The way to store binary information into DNA based medium storage and then retrieve it back is called end-to-end DNA storage. It has several steps as seen on Figure 3.2.

- **Encoding**: Transform any kind of infromation into a nucleotide stream (A, C, G or T);
- **Synthesis**: Synthesize the nucleotide stream into an actual DNA molecule;
- **Encapsulation**: Encapsulate the DNA molecule into a DNA medium for long term storage;
- **Thermal damage**: The DNA medium can be damaged by thermal damage, this can cause the DNA to break or be damaged during the storage process;
- **Release**: The DNA molecules can be released from the medium, it gives us the ability to retrieve the information back.
- **Sequencing**: The DNA molecules are sequenced on a computer, this allows us to digitally retrieve the information back as DNA sequences.
- **Decoding**: The DNA sequences are decoded back into the original information.

3.1.4 Constraints

Unfortunately, nucleic acids have biological constraints and can not be assembled in any order like it is the case for binary digits. The DNA strands have to be created in a way that the double helix binds well together and is not immediately desctructed. We must therefore respect the biological constraints to build strong strands that can last for a certain period of time.

In this part, we are going to go through some of the constraints that we have to respect to build a DNA strand and be able to recover it when sequencing it. Unfortunately, the list of constraints is not exhaustive and in the real life, each arangement of nucleic acids has an impact on the strength of a strand, therefore we can only simulate the longevity of a strand thanks to the actual discoveries but not strictly respect the biological constraints.

All constraints could be reduced to limitations regarding GC content, long strands of a single nucleotide (so-called homopolymers), several repeated subsequences in a strand and motifs with biological relevance. In the next sections, we are going to divide the constraints into each step of the process, the explanation for each constraint comes directly from [6].

Synthesis

For example, to synthesis synthetic DNA, *in silico* designed constructs have to be split into smaller fragments [usually 200–3000 base pairs (bp)] [7]. The fragments are then splitted into several oligonucleotides (so-called oligos) [usually 40-100 bp] that are individually synthetized. Once synthetized, all oligos are merged back together with either ligase or polymerase-based methods. One of the constraints on the GC content comes from the fact that depending on the synthetis method and the overall GC content of a fragment, the GC content of each oligo has to be within a specific range. In oligos with high GC content, neighboring guanines tend to form an increased amount of hydrogen bonds, leading to inter/intra-strand folding [8]. To assemble oligos into larger fragments, the melting temperature (and thus the GC content) should only deviate slightly between oligos. To adhere to this restriction, the designed DNA fragments should be homogenous with respect to GC content. Homopolymers further increase the synthesis complexity, leading to fragments that are only synthesizable by using modified oligos and more sophisticated assembly methods, resulting in increased synthesis costs.

PCR: Polymerase Chain Reaction

The amplification of DNA using polymerase chain reaction (PCR) is indispensable for biological science. From DNA synthesis over cloning to DNA sequencing, PCR is used in a wide range of applications. One important factor of a successful PCR is the base composition of the amplification substrate. High melting temperatures due to high GC content of the DNA fragments hinder the separation of strands during the denaturation phase of the PCR. This reduces the yield of the PCR process, since the polymerase cannot efficiently synthesize the growing strand in the presence of previously existing hydrogen bonds. Stretches of repetitive DNA or high GC content can lead to the formation of secondary structures, hindering the elongation of the growing strand. Repetitive regions, as well as homopolymers, can also lead to polymerase slippage, a process in which polymerase briefly loses the connection to the template strand and reconnects at a different position with similar nucleotides content [9].

Storage

Further restrictions on the composition of the DNA construct are due to the cloning process: the GC content should be close to the GC percentage of the host genome and motifs used for the cloning process have to be avoided during the design of the DNA construct

Sequencing

The base composition of a DNA fragment is also an important factor for the successful retrieval of genetic information using DNA sequencing technologies Illumina sequencing, Oxford Nanopore and PacBio sequencing technologies are biased toward DNA with an intermediate GC content, leading to reduced coverage of regions with strongly deviating GC content [10]. Illumina and Nanopore sequencers also show an increased error rate in the presence of homopolymers [10]. Depending on the sequencing method used, the resulting data show increased substitution rates for specific DNA patterns: for PacBio data, common substitution

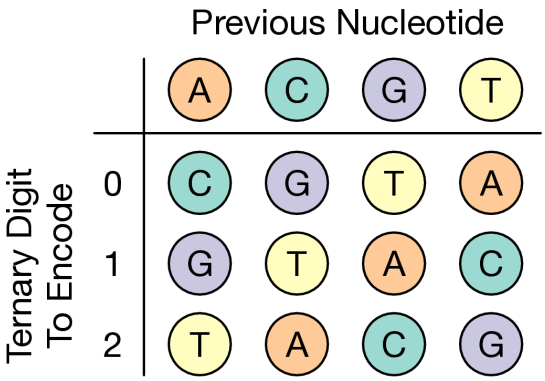

Figure 3.3: Encoding of the trit produced by a 3-ary Huffman tree for each symbol

patterns are CG → CA and CG → TG, Nanopore data contain an increased amount of TAG → TGG and TAC → TGC substitutions [11] and a common substitution pattern in Illumina data is GGG → GGT [12].

3.1.5 MESA: Mosla Error Simulator

In order to simulate without going through an expensive and long process that is DNA synthesis and sequencing, we are going to use a simulator that takes into account a large majority of biological constraints. This simulator has been introduced [6] in March 2020 and is a web application for the assessment of DNA fragments in terms of guanine-cytosine (GC) content, homopolymer occurrences and length, repeating subsequences and undesirable sequence motifs. Furthermore, MESA contains a mutation simulator, using either the error probabilities of the assessment calculation, literature-based or user-defined error rates and error spectra. MESA is fully customizable using an easy-to-use web interface, without requiring programming experience. All functionality of MESA is also contained in a REST API, enabling the incorporation of MESA evaluations into custom workflows for high-throughput evaluation and error simulation of DNA.

As we have seen in the previous section, DNA has a lot of constraints during the synthesis, storage, PCR and sequencing step; With this simulator it is now possible to have a feedback of the strength of a particular DNA strand and thus help us to move towards the best DNA coding to ensure good information retrieval in the end.

3.2 DNA coding

3.2.1 Goldman coding

In 2013, Goldman and al. [13] proposed an algorithm to encode digital data into binary while respecting the main constraint of DNA sequencing, homopolymers. The technique proposes to encode data first in a 3-ary Huffman tree so that each byte can be encoded into the digits $0, 1, 2$ giving a smaller representation to more frequent data and longer one to less frequent data. Once each byte has its own code representation in the tree, the digits can be encoded by mapping to one of the nucleotides while ensuring that the previous encoded nucleotides is not use to encode the current digit. A representation is shown on the Figure 3.3 for the mapping of the digits regarding the last encoded nucleotide.

3.2.2 PAIRCODE

PAIRCODE [14] is a technique proposed in 2019 to encode any symbols (it is not restricted to binary data) to fixed length codewords. The quaternary constrained codeword $\mathcal{C}*$ is constructed using mainly pair-elements from the following dictionary:

$$\mathcal{C}_1 = \{AT, AC, AG, TA, TC, TG, CA, CT, GA, GT\}$$

It is possible to build any codeword of even length by concatening words from these 10 elements. To build codeword of odd length, it is possible to concatenate one of the word from the dictionary:

$$\mathcal{C}_2 = \{A, C, G, T\}$$

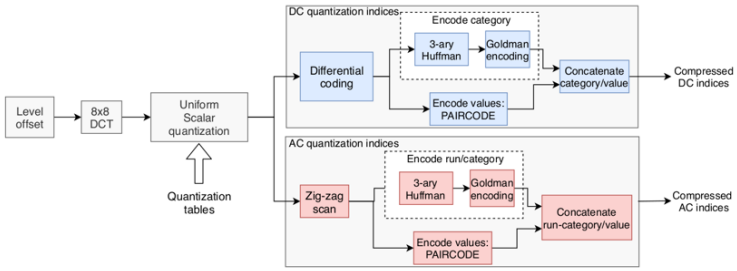

Figure 3.4: Schema of the JPEG DNA codec

This construction guarantees that the final encoded DNA stream does not contain homopylmers and a GC-percentage higher than the AT-percentage.

3.2.3 JPEG DNA codec

The purpose of the JPEG DNA codec is to encode a raw image to DNA code using the JPEG codec and encoding the coefficients to actual DNA nucleotides thanks to the PAIRCODE and Goldman coding.

Before turning an image into a DNA stream, the first steps are to extract the DCT coefficients that are going to be encoded.

With a raw image as input of shape either H×W (gray image) or H×W×3 (RGB image), the first step is to shift the values of the image by -128 to center it around 0 since the values of the pixels are encoded on *uint8* values and are thus betweem 0 and 255. Then the discrete cosine transform is applied on these centered values to extract the (8×8) block DCT coefficients. Next, since the codec is dealing with visual information, a uniform scalar quantization is used to turn continuous values to discrete coefficients thanks to predefined quantization tables that give more importance to low frequency cosines (because the human eye is more sensitive to low frequency signals than high frequency ones). The quantization introduced a lot of zero for high frequencies on the lower right part of the block. The last step is then to flatten the block DCT quantized coefficients in a zigzag maneer starting from the upper left corner, this way, there will be a large part of zeros at the end of the 64-long flatten array.

The flatten array is now composed of the DC value (constant intensity) as first element and then 63 AC values corresponding to each particular cosine frequency. As we will see, the DC and AC values are encoded into DNA in two different ways.

In the classical JPEG workflow, each category is mapped to a binary representation thanks to the Huffman tree that will encode the most frequent AC/DC categories to the shortest words. For the DNA version, they used a 3-ary Huffman tree to transform the category into trits and then encode these categories with the Goldman algorithm, the category are determined by the range in which the value falls. The AC and DC indices are then encoded using a fixed length l codebook that belongs to a code which has been generated using PAIRCODE.

For example, here is how the absolute AC values are categorized:

- $\{0\} \to 0$
- $[1, 5] \to 1$
- $[6, 17] \to 2$
- $[8, 82] \to 3$
- $[83, 375] \to 4$
- $[376, 1263] \to 5$
- $[1264, 5262] \to 6$
- $[5263, 17579] \to 7$

And for each category, we have a fixed number of codewords with fixed length in a specific codebook. Here is the example of the codebooks used in the JPEG DNA codec algorithm per category (as shown on Figure A.2):

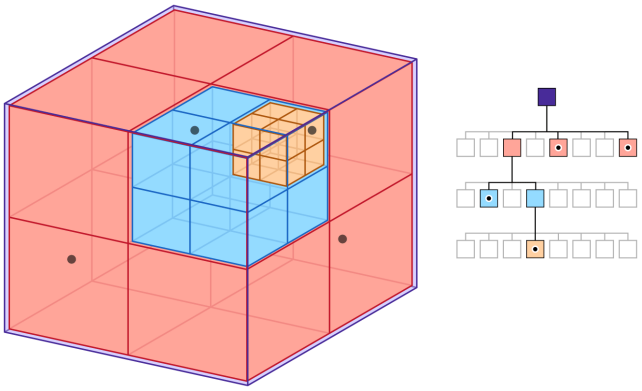

Figure 3.5: Octree representation of a point cloud

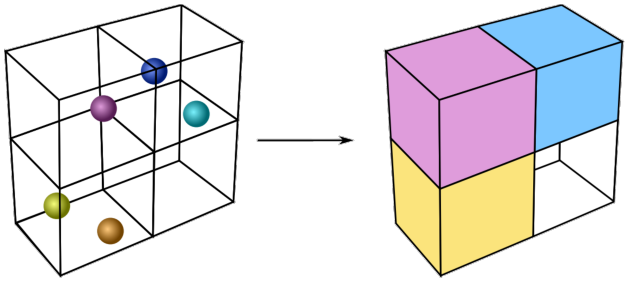

Figure 3.6: Voxelized representation of a point cloud

- $0 \rightarrow 10$ words of length 2
- $1 \rightarrow 24$ words of length 3
- $2 \rightarrow 130$ words of length 4
- $3 \rightarrow 586$ words of length 5
- $4 \rightarrow 1776$ words of length 6
- $5 \rightarrow 7998$ words of length 7
- $6 \rightarrow 24634$ words of length 8
- $7 \rightarrow 110660$ words of length 9

3.3 Point cloud compression

Numerous methods have been proposed to compress point clouds in the literature. They are usually based on different structures than a classical list of coordinates. Octrees, for example, have been widely used for this purpose [15]. The octree representation consists of diving recursively the three dimensional space as nodes of a tree as shown on Figure 3.5.

Compression algorithms using learning based autoencoders architectures have also demonstrated good performance. While some take as input point coordinates [16], others take as input voxelized versions of the point clouds. Voxelized point clouds consist of occupancy grid of regular spaced points so that several points are merged together in a single voxel as shown on Figure 3.6. A voxel is similar to a three dimensional pixel. These three dimensional grids can be then used as input for a *3D convolutional layer*.

The current state of the art for point cloud compression that we will use in this project is a model called "Latent Space Slicing For Enhanced Entropy Modeling in Learning-Based Point Cloud Geometry Compression" and that has been developed by Nicolas Frank, Davi Lazzarotto and Touradj Ebrahimi at the MMSPG laboratory (EPFL).

3.3.1 Model architecture

This model is shown on Figure 3.7 and consists in a 3D autoencoder architecture with latent entropy coding. The input of the model is an occupancy cubic grid with $k \times k \times k$ voxels represented by 1 when occupied and 0 otherwise.

The first block (Analysis transform) of the model is composed of 3 *3D convolutional layers* and 2 *convolutional residual blocks* arranged staggered which produces a latent representation y of shape $l \times l \times l \times d$ with d being the latent dimension.

This latent representation is then fed into an Hyper-Analysis transform block yielding

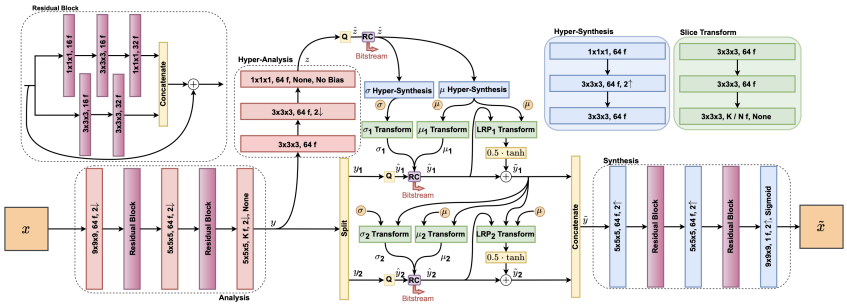

Figure 3.7: The "Latent Space Slicing For Enhanced Entropy Modeling in Learning-Based Point Cloud Geometry Compression" architecture with 2 slices.

z. This hyperprior is passed to the bitstream as side information after quantization, and is used to model the entropy of the quantized latent features $\hat{y}$ after going through the hyper-synthesis.

While in other solutions for learning-based point cloud compression the hyperprior would be the only variable used to estimate the scale and mean of $\hat{y}$, they use previously decoded channels for entropy modeling.

The latent representation y is sliced along the channel dimension into N non-overlapping and equally sized tensors y_i with $i \in \{1, \ldots, N\}$.

Once the latent representation y is sliced, they compute the entropy parameters (μ_i, σ_i) for each slice y_i from the global entropy parameters (μ, σ) and the previously decoded latent representation slices $\tilde{y}_j \, \forall j \in \{0, \ldots, i-1\}$.

The final latent representation reconstruction $\tilde{y}_i$ is produced from $\hat{y}_i$ after going through a latent residual prediction (LRP) transform that predicts the quantization error $y_i - \hat{y}_i$ in order to take into account this error from the global entropy parameters (μ, σ), the current $\hat{y}_i$ and previously decoded latent representation slices $\tilde{y}_j \, \forall j \in \{0, \ldots, i\}$. A *tanh* non-linearity scaled by a factor 0.5 is applied to the output of the transform to keep the output of the LRP within the range of quantization error. The predicted residuals are then added to $\hat{y}_i$, generating $\tilde{y}_i$. These slices are then concatenated along the channel dimension before going through the synthesis transform. This last learned block finally generates the output block $\tilde{x}$ containing a probability estimation for the occupancy of each voxel in the grid.

They trained the model from end-to-end between the input occupancy grid and output probability occupancy grid as a rate-distortion minimization problem represented by the loss function expressed as $\mathcal{L} = R + \lambda D$. The trade-off parameter λ is used to balance the importance of compression rate against reconstruction quality. The distortion is computed from a focal loss (FL) which can be expressed as:

$$\mathrm{FL}(x, \tilde{x}) = -x\alpha(1-\tilde{x})^{\gamma}\log(\tilde{x}) - (1-x)(1-\alpha)\tilde{x}^{\gamma}\log(1-\tilde{x})$$

with α and γ being configurable hyperparameters. This focal loss is a so-called binary weighted focal crossentropy loss.

The last step to recover the actual binary occupancy grid for each voxel from the output probability occupancy grid $\tilde{x}$ is to apply a threshold t and round every value above t to 1 and 0 otherwise. While a naive approach would be to set this threshold t to 0.5, they decided to find the best threshold $t \in [0, 1]$ such that it minimizes the point-to-point MSE [17] between the original and reconstructed rounded block.

This technique was introduced by [18] and has proven to enhance reconstruction quality without significantly impacting the bitrate as it is only required to add a threshold $t \in [0, 100]$ in the final bitstream.

Chapter 4

Implementation

In this section, we want to build a complete pipeline from a raw point cloud to a DNA strand that will be later synthestized in a medium and kept for a long period (Could be dozen or even hundred of years). After this long period of time, we would like to recover our point cloud as faithful as possible to the original.

We thus have all the building blocks to construct this pipeline and be able to encode and decode point clouds. However, this pipeline will not be constructed to be the most efficient and optimized DNA code point cloud compression algorithm but it will be a baseline on which we can base ourselves to compare future models.

4.1 Point cloud latent representation

The first step is to turn the point cloud into a smaller latent representation. For this purpose, we can use the point cloud compression model previously described in section 3.3 that has already learned the principal features of point clouds.

The latent representation y contains a compressed representation with less information but still all the necessary information to reconstruct the original point cloud. From a block of shape $k \times k \times k$, once passed through the Analysis Transform, the latent reprsentation y has a shape $\lceil \frac{k}{8} \rceil \times \lceil \frac{k}{8} \rceil \times \lceil \frac{k}{8} \rceil \times d$ with d being the arbitrary latent depth. Therefore if the latent depth d is well chosen, the latent representation contains all information to retrieve the original point cloud while being smaller.

In our particular implementation, we chose a latent depth of 160 which means that we indeed have a smaller shape since $8 \times 8 \times 8 \geq 160$.

We also have to make a choice on which model to use for the transfer learning of the weights since several models are available for different bitrate/distortion tradeoffs. As we will see in the next sections, the nucleotiderate mostly depends on the input shape without impacting the distortion no matter which one is chosen. Thus to get the best reconstruction possible, we will choose the highest λ value (1750) for the model. The λ value is weighting the focal loss in the global loss used to train the model as explained in Section 3.3.

Now that we have a latent representation, we have to go to the next step, which is the JPEG DNA codec described in section **??**, but this codec has some requierements that have to be met in order to achieve the best quality, nucleotide rate, all these details will be discuss in the next section.

4.2 Latent representation with JPEG DNA codec

In this part, we will use the JPEG DNA codec but not the full one, we will drop all the part related to JPEG and thus encode directly the coefficients that are normally produced by the last DCT transform. So the input of the codec will be quantized discrete coefficients. To meet these requirements, we will have to change few things from the latent representation y.

4.2.1 Dimensionality

The output of the Analysis block is a $l \times l \times l \times d$ latent representation y but since the codec is built for image purpose, it only accepts an image as input which is either of shape

$H \times W$ (gray image) or $H \times W \times 3$ (RGB image). Consequently, we have to tweak our latent representation y so that it satisfies this requirement, we have several possibilities for that, I tried two different approaches:

- The first approach is to merge the two inner dimension together, in that case we have a $l \times l^2 \times d$ shape. We can then encode several $l \times l^2$ images alogn the latent features dimension. This approach allows to process each feature "independently";
- The second approach is more simple and consists of merging the three first dimensions together, hence we end up with a $l^3 \times d$ image. We can thus encode directly the full image with the codec.

Each approach has its own pros and cons.

The first one aims at treating each feature separately so that we can encode regarding each feature distribution and hope for less quantized values and in the end, a smaller number of oligos. Although, when encoding the final flat nucleotide stream, we have to add the length of each latent oligo length which is avoidable.

The second as for it, is easier to compute since it is a single gray image like, so we have in the end directly a flat nucleotide stream for a block.

Finally, as the first one does not offer better performance while enlarging the final stream length (and thus impact the final nucleotiderate), we will simply merge the first three dimensions (dimensions of the feature block) as proposed in the second approach to produce a 2D tensor compatible with the codec.

4.2.2 Quantization

The other important aspect of the codec is the value type. The Analysis block produces continuous values $y \in [-\inf, \inf]$ due to the nature of convolutional layer and the linear activation function of the last layer but the codec is built to encode discrete integer values as it is the case for the DCT coefficients. This means that we have to convert the latent representation y to an integer value type while ensuring the y range is mapped to a range $[0, span]$ as a bijective maneer. Here the $span$ value is fixed value chosen at compression time as a quality parameter. It is used as a nucleotiderate/distortion tradeoff. The greater the $span$ value, the better the quality and the greater the nucleotiderate. Contrastly, the lower the $span$ value, the worse the quality and the lower the nucleotiderate.

To implement such a quantization transform, we can naively do a linear mapping between $[\min y, \max y]$ to $[0, span]$. With this transform, we create $span + 1$ ranges of equal length $\frac{\max y - \min y}{span}$ in which the y values will fall. It assumes that y is uniformly distributed because it gives the same range length in which to fall for each y value.

Finally, to get the quantized integer value, we can simply round the final mapped values and it will gives us at most $span + 1$ integer values to encode with the JPEG DNA codec.

The mapping can simply be described as:

$$z = \text{round}\left(span \frac{y - \min y}{\max y - \min y}\right)$$

.

4.2.3 DCT

In the original JPEG codec, each 8×8 block of the images are first centered around 0 (simply shift by -128) and then transformed into the 8×8 DCT values before being quantized with a default perceptual quantization table. For our purposes, we will use a custom way to do the DCT that is more adapted to our needs considering that we already have quantized coefficients.

Because they are built for perceptual fidelity, we will not be dividing by quantization tables and thus will not have the long tail of 0's after the zigzag transform that normally appears for high frequencies DCT coefficients. All coefficients will most likely not be 0, so we will have to encode them all.

All these transforms can be seen as a block represented on Figure 4.1 that first reshape the 4 dimensional tensor into a 2 dimensional one and then quantize it with the approach seen above.

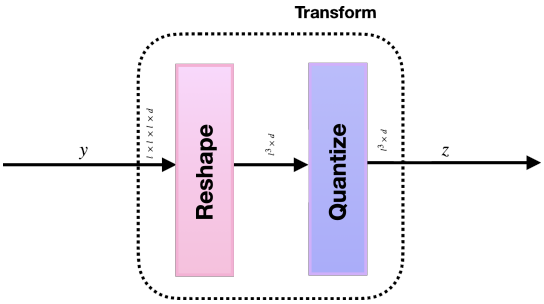

Figure 4.1: Reshape + quantization layers

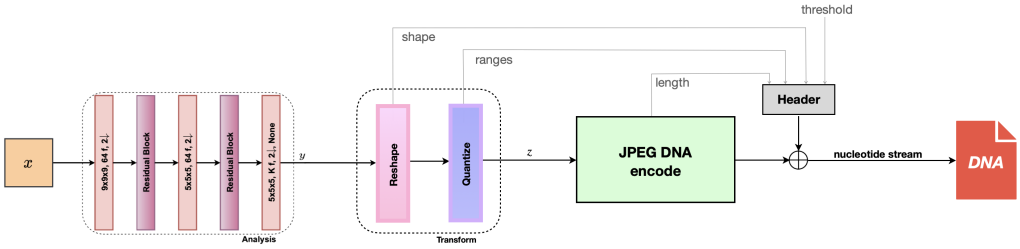

Figure 4.2: The full end-to-end mode that turns voxelized point clouds into a DNA stream

4.3 Nucleotide stream

We now have a full pipeline that can be used to encode and decode point clouds. The last step is to produce the actual nucleotide stream from the output of the codec and all intermiediate information that are needed in order to fully decode the stream.

The intermediate informations needed to produce the nucleotide stream are:

- The threshold used to turn the reconstructed block $\tilde{x}$ (the probability occupancy grid described in 3.3.1) as an *uint8* value;
- The oligo length used by the JPEG DNA codec (by default set to 200);
- The quantization range used for the latent representation y as described in 4.2.2;
- The shape of the latent representation y in order to reshape the decoded array of the codec since we merge all inner dimensions together to encode with the codec;

To turn a byte array into a nucleotide stream, we naively assume that each two consecutive bits can be considered as a nucleotide. We can use the mapping $00 \rightarrow$ A, $01 \rightarrow$ C, $10 \rightarrow$ G, $11 \rightarrow$ T. With this technique we can produce a 4 nucleotide stream for each byte of the byte array.

Here is how many nucleotides we can produce to represent our intermerdiate informations:

- The threshold is a single byte since it is a *uint8* value $\in [0, 100]$ so we can produce 4 nucleotides;
- The oligo length is also encoded in a single byte, it thus produces 4 nucleotides as well;
- The quantization range are represented as two *float32* values, each value is encoded on 4 bytes so we can produce a total of 32 nucleotides for the quantization range;
- The shape of the latent representation has the form $l \times l \times l \times d$ and each dimension is encoded on a single byte so we can produce a total of 16 nucleotides for the shape;

The final stream is obtained by concatenating all the additional informations and the nucleotide stream. In the end we have a total of $4 + 4 + 32 + 16 + n$ nucleotides where n is the number of nucleotides produced by the codec.

4.4 End-to-end model

With all theses changes, we now have a fully working end-to-end model capable of compressing a voxelized point cloud into a DNA stream respecting some constraints (homopolymers and

kmer) thanks to PAIRCODE and the Goldman encoding. The model is shown on Figure 4.2.

4.5 Reconstruction

Starting from the nucleotide stream, we can fully reconstruct the point doing by doing all steps in reverse order. First, we extract all the additional informations and the codec stream from the nucleotide stream. Then, we can decode the codec stream to reconstruct the latent representation $\tilde{y}$ and reshape it to the shape that was encoded in the stream. Then, we can decode the latent representation $\tilde{y}$ using the dequantization transform with the quantization range that was also encoded in the stream in order to obtained the reconstrcuted block $\tilde{x}$ Finally, we can round the probabibility occupancy grid $\tilde{x}$ thanks to the threshold that was encoded in the stream and we can obtain the reconstructed binary occupancy grid.

Bibliography

[1] A. Dellinger, "The environmental impact of data storage is more than you think — and it's only getting worse," jun 2019. https://www.mic.com/impact/the-environmental-impact-of-data-storage-is-more-than-you-think-its-only-getting-worse-18017662.

[2] J. Bohannon, "Dna: The ultimate hard drive," aug 2012. https://www.wired.com/2012/08/dna-data-storage.

[3] Y. Erlich and D. Zielinski, "DNA Fountain enables a robust and efficient storage architecture," Mar. 2017.

[4] K. Matange, J. M. Tuck, and A. J. Keung, "Dna stability: a central design consideration for dna data storage systems," *Nature communications*, vol. 12, p. 1358, March 2021.

[5] K. B. Mullis, "The unusual origin of the polymerase chain reaction," *Scientific American*, vol. 262, p. 56–65, Apr 1990.

[6] M. Schwarz, M. Welzel, T. Kabdullayeva, A. Becker, B. Freisleben, and D. Heider, "MESA: automated assessment of synthetic DNA fragments and simulation of DNA synthesis, storage, sequencing and PCR errors," *Bioinformatics*, vol. 36, pp. 3322–3326, 03 2020.

[7] S. Kosuri and G. Church, "Large-scale de novo dna synthesis: Technologies and applications," *Nature methods*, vol. 11, pp. 499–507, 04 2014.

[8] M. Jensen, M. Fukushima, and R. Davis, "Dmso and betaine greatly improve amplification of gc-rich constructs in de novo synthesis," *PloS one*, vol. 5, p. e11024, 06 2010.

[9] A. Fazekas, R. Steeves, and S. Newmaster, "Improving sequencing quality from pcr products containing long mononucleotide repeats," *BioTechniques*, vol. 48, pp. 277–85, 04 2010.

[10] D. Laehnemann, A. Borkhardt, and A. McHardy, "Denoising dna deep sequencing data–high-throughput sequencing errors and their correction," *Briefings in bioinformatics*, vol. 17, 05 2015.

[11] J. Weirather, M. d. Cesare, Y. Wang, P. Piazza, V. Sebastiano, X.-J. Wang, D. Buck, and K. Au, "Comprehensive comparison of pacific biosciences and oxford nanopore technologies and their applications to transcriptome analysis," *F1000Research*, vol. 6, p. 100, 06 2017.

[12] M. Schirmer, R. D'Amore, U. Ijaz, N. Hall, and C. Quince, "Illumina error profiles: Resolving fine-scale variation in metagenomic sequencing data," *BMC Bioinformatics*, vol. 17, 03 2016.

[13] N. Goldman, P. Bertone, S. Chen, C. Dessimoz, E. M. LeProust, B. Sipos, and E. Birney, "Towards practical, high-capacity, low-maintenance information storage in synthesized dna," *Nature*, vol. 494, pp. 77–80, Feb 2013. 23354052[pmid].

[14] M. Dimopoulou, M. Antonini, P. Barbry, and R. Appuswamy, "A biologically constrained encoding solution for long-term storage of images onto synthetic dna," in *2019 27th European Signal Processing Conference (EUSIPCO)*, pp. 1–5, Sep. 2019.

[15] J. Kammerl, N. Blodow, R. Rusu, S. Gedikli, M. Beetz, and E. Steinbach, "Real-time compression of point cloud streams," in *Proceedings - IEEE International Conference on Robotics and Automation*, 05 2012.

[16] X. Wen, X. Wang, J. Hou, L. Ma, Y. Zhou, and J. Jiang, "Lossy geometry compression of 3d point cloud data via an adaptive octree-guided network," in *2020 IEEE International Conference on Multimedia and Expo (ICME)*, pp. 1–6, 2020.

[17] D. Tian, H. Ochimizu, C. Feng, R. Cohen, and A. Vetro, "Geometric distortion metrics for point cloud compression," in *2017 IEEE International Conference on Image Processing (ICIP)*, pp. 3460–3464, 2017.

[18] M. Quach, G. Valenzise, and F. Dufaux, "Improved deep point cloud geometry compression," in *2020 IEEE 22nd International Workshop on Multimedia Signal Processing (MMSP)*, pp. 1–6, 2020.

Appendix A

Appendix

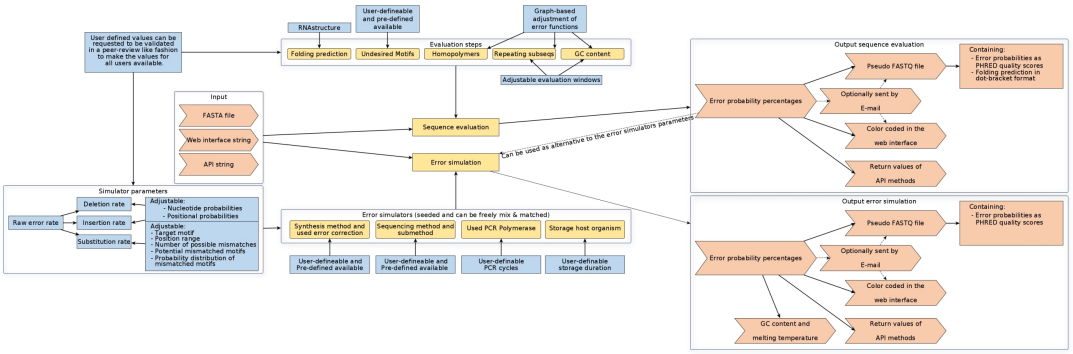

Figure A.1: Mesa workflow

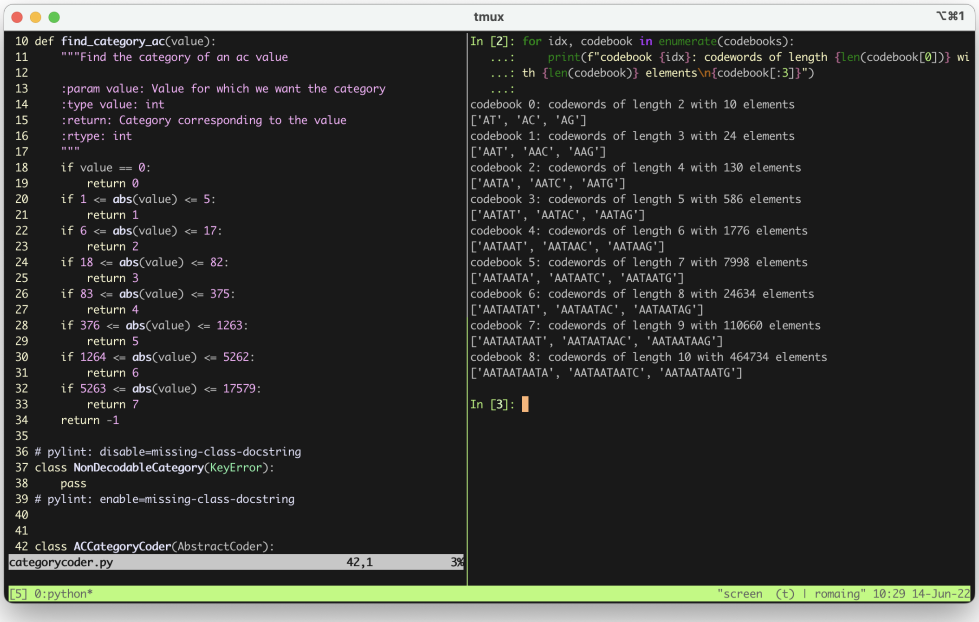

Figure A.2: The codebooks used by the JPEG DNA codec