Entropy–Rank Ratio: A Novel Entropy–Based Perspective for DNA Complexity and Classification

Emmanuel Pio Pastore[1], Giuseppe Passarino[1], Peppino Sapia[2], Francesco De Rango[1*]

[1]Department of Biology, Ecology and Earth Science
University of Calabria, 87036 Rende, Italy
[2]Department of Mathematics and Computer Science
University of Calabria, 87036 Rende, Italy

November 06 2025

Abstract

Shannon entropy is widely used to measure the complexity of DNA sequences but suffers from saturation effects that limit its discriminative power for long uniform segments. We introduce a novel metric, the *entropy rank ratio* R, which positions a target sequence within the full distribution of all possible sequences of the same length by computing the proportion of sequences that have an entropy value equal to or lower than that of the target. In other words, R expresses the relative position of a sequence within the global entropy spectrum, assigning values close to 0 for highly ordered sequences and close to 1 for highly disordered ones. DNA sequences are partitioned into fixed-length subsequences and non-overlapping n-mer groups; frequency vectors become ordered integer partitions and a combinatorial framework is used to derive the complete entropy distribution. Unlike classical measures, R is a normalized, distribution-aware measure bounded in $[0, 1]$ at fixed (T, n), which avoids saturation to $\log_2 4$ and makes values comparable across sequences under the same settings. We integrate R into data augmentation for convolutional neural networks by proposing ratio-guided cropping techniques and benchmark them against random, entropy-based, and compression-based methods. On two independent datasets, viral genes and human genes with polynucleotide expansions, models augmented via R achieve substantial gains in classification accuracy using extremely lightweight architectures.

1 Introduction

DNA is the fundamental molecule of life and is found in all living organisms. It represents a code formed by four distinct letters: the nitrogenous bases (Adenine, Cytosine, Guanine, Thymine). These bases, by combining in sequences of varying lengths and complexity, give rise to highly variable genetic information[1]. In fact, it has been known since the last century that DNA sequences encoding proteins are read in groups of three consecutive letters; in total, 64 triplets encode 20 amino acids (as well as the start and stop signals of the code)[2].

Since the discovery of this coding system, efforts have been made to represent the information and complexity of DNA sequences in a mathematical and information-theoretic perspective[3, 4]. Since the 1940s, mathematical and algebraic techniques have been applied to this subject, but it was only after the discovery of the structure of DNA and the introduction of Shannon entropy that it became theoretically possible to calculate the entropy of DNA sequences[4, 5, 6]. This development sparked strong interest in the topic during the 1980s and 1990s[4, 6, 7], aided by the increasing availability of computational resources. However, several challenges arise when calculating the Shannon entropy

*Corresponding author: francesco.derango@unical.it

of sequences: without additional precautions, the entropy of sufficiently long sequences converges to 2 bits, and the practical applications of entropy itself are not immediately evident.

Entropy-based techniques have, however, been widely used in Machine Learning for a long time[8, 9], and have also been applied in DNA sequence classifiers[10, 11]. Neural networks commonly used in this regard are CNNs (Convolutional Neural Networks), which capture local sequence patterns and aggregate them into global representations[12, 13].

The objective of this paper is to develop a new method for characterizing the complexity of DNA sequences and to integrate it with the aforementioned neural network architectures in order to significantly improve precision and performance on datasets of viral genes from a GitHub repository[14], and of human genes associated with polynucleotide expansion[15].

2 Materials and Methods

DNA can be viewed as an alphabet [6, 16, 17, 18] $\mathcal{A}$ consisting of $\lambda_1 = 4$ different single-digit letters

$$\mathcal{A} = \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}\}. \tag{1}$$

Definition 2.1. *Let $V = \mathbb{R}^{\lambda_1}$ be the set of lists (vectors) whose components are indexed by the alphabet $\mathcal{A} = \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}\}$.*

Any DNA sequence w of length L is represented by a frequency vector $x \in V$. Here, length(x) *denotes the number of components in the frequency vector x corresponding to the counts of each letter.*

Let $x_j \in \mathcal{A}$ for $j = 1, \ldots, \lambda_1$ be a letter of the alphabet. Every DNA sequence can be represented as a frequency vector

$$x = (a_1,\, a_2,\, a_3,\, a_4), \tag{2}$$

where a_j denotes the number of occurrences of the letter x_j in the sequence w. By convention, we order the components such that

$$a_1 \geq a_2 \geq a_3 \geq a_4 \geq 0. \tag{3}$$

Note: *This ordering is adopted solely for computational efficiency; it loses the original correspondence between each count and its specific nucleotide (or n-tuple), which is irrelevant to the calculation of Shannon entropy. In our representation, it is also important to include components with zero count to avoid confounding later definitions.*

Definition 2.2. *From Definition 2.1, we have that the frequency vector x has λ_1 components, i.e.,*

$$length(x) = \lambda_1 \tag{4}$$

[19]

Definition 2.3. *We can generalize the alphabet by grouping letters into sequences of length n, called n-*tuples*[20]. The alphabet for n-tuples is denoted by $\mathcal{A}_n$, where each element $x_j \in \mathcal{A}_n$ is an n-tuple derived from $\mathcal{A}$.*

Definition 2.4. *Since each position in an n-tuple can be any of the 4 letters, the size (cardinality) of the alphabet for n-tuples is*

$$\lambda_n = 4^n \tag{5}$$

Thus, by Definition 2.3, the frequency vector x has

$$length(x) = 4^n \tag{6}$$

Accordingly, a sequence w can be represented by the vector

$$x = (a_1, a_2, \ldots, a_{\lambda_n}), \tag{7}$$

where a_j denotes the occurrence count of the n-tuple x_j. We conventionally order these counts in non-increasing order:

$$a_1 \geq a_2 \geq \cdots \geq a_{\lambda_n} \geq 0. \tag{8}$$

(Compare with the ordering in Definition 2.1.)

Note: *For the sake of notational simplicity, we shall denote $\lambda_n = 4^n$ simply as λ. Hence, throughout the paper we implicitly assume $\lambda = \lambda_n = 4^n$; whenever n changes, so does λ. In particular, for $n = 1$ we have $\lambda = 4$.*

Definition 2.5. ***(Case** $n = 1,\ L > 0$**)** Let*

$$b_j = \frac{a_j}{L} \tag{9}$$

be the relative frequency of the letter x_j when $n = 1$*, so that*

$$\sum_{j=1}^{\lambda} a_j = L, \tag{10}$$

and hence

$$\sum_{j=1}^{\lambda} b_j = 1. \tag{11}$$

***(Case** $n > 1,\ L \geq n$**)** When $n > 1$ and the sequence w is divided into non-overlapping n-tuples, let*

$$M = \left\lfloor \frac{L}{n} \right\rfloor \quad \textit{and define} \quad b_j = \frac{a_j}{M}. \tag{12}$$

This is the normalized frequency of the letter or group of letters (n-tuple). Hence

$$\sum_{j=1}^{\lambda} a_j = M, \quad \textit{and} \quad \sum_{j=1}^{\lambda} b_j = 1. \tag{13}$$

Definition 2.6. *A sequence w can be divided into ordered n-tuples, indexed by m, indicating the position of the n-tuple in the sequence. By counting these n-tuples, we determine the occurrences a_j of each n-tuple type x_j, which form the components of the vector x.*

Definition 2.7. *Consider a sequence $w = (w_1, w_2, \ldots, w_L)$ and let x_i^m denote the i-th letter in the m-th n-tuple. The set of n-tuples is said to be* non-overlapping *if each letter in w is assigned to exactly one n-tuple. This is achieved by defining the correspondence*

$$w_k \quad \textit{with} \quad k = n(m-1) + i, \quad \textit{for } i = 1, \ldots, n, m = 1, \ldots, M \tag{14}$$

where M is the total number of non-overlapping n-tuples.

Definition 2.8. *Here, $\lfloor \cdot \rfloor$ denotes the integer part of the division. Note that, without such operation, M is an integer if and only if L is a multiple of n. Otherwise, there will be a remainder of letters given by*

$$z = L - n\,M = L \bmod n. \tag{15}$$

Definition 2.9. *The letters w_{L-z+1} to w_L (when $z \neq 0$) are not included in any n-tuple and are therefore ignored in subsequent calculations.*

We can also divide the sequence w of length L into N equal subsequences[21] w_t, each of length T, where

$$N = \left\lfloor \frac{L}{T} \right\rfloor . \tag{16}$$

2.1 Shannon Entropy of a DNA Sequence w

To measure the entropy of a DNA sequence, one must first choose the method of assigning an entropy value[22, 23]. For example, one could refer to thermodynamic entropy[23], topological entropy[6, 17], or Shannon entropy[5], among others. In this context, Shannon entropy will be used.

The classical Shannon entropy of a sequence w, denoted $S(w)$, is defined (based on previous definitions) as follows:

$$S(w) = -\sum_{j=1}^{\lambda} b_j \log(b_j), \tag{17}$$

where, by convention, log denotes the base-2 logarithm.

Definition 2.10. *If $b_j = 0$, we set $b_j \log(b_j) = 0$ in the summation to represent the fact that this element is not counted in the calculation of Shannon entropy. This convention is useful for maintaining vectors with a consistent number of elements.*

Here, Shannon entropy is essentially a measure of the informational complexity (in bits) of an alphabetical sequence[6].

Now, we address a previously overlooked question: why introduce a generalized alphabet of n-tuples?

The answer is straightforward: for very long DNA sequences generated by a uniform i.i.d. process, the distribution of letters becomes uniform, leading to

$$S(w)_{\text{with } L \gg 1} = -\sum_{j=1}^{4} b_j \log(b_j) \;\rightarrow\; \log(4) \;=\; 2, \tag{18}$$

since for $L \gg 1$ (and $n = 1$) $b_j \simeq 0.25$ for all $1 \leq j \leq 4$[7].

Generally, for very long sequences and very short non-overlapping n-tuples generated by a uniform process:

$$S(w)_{\text{with } L \gg 1} = -\sum_{j=1}^{\lambda} b_j \log(b_j) \rightarrow \log(\lambda). \tag{19}$$

However, this issue becomes less pronounced for short n-tuples. Conversely, if we increase the length of the tuples, for example when $n \rightarrow L$, the entropy $S \rightarrow 0$ because very few n-tuples will be represented among the total possible λ n-tuples.

A further method, that distinguishes our approach, is the assignment of an *average* entropy value[24] (which we will be referring to as S_w throughout the rest of the paper to avoid confusion with the classical case) to a sequence w. This value is defined as the average of the entropies S_w^t of the t-th subsequences w_t of w. This approach is computationally more efficient and avoids the aforementioned problem of saturating into the maximum possible entropy.

In the following theorem, the index will be understood as that of a subsequence within a sequence.

Theorem 2.1 (Concatenation with exact bound)**.** *Fix T, n and let $\lambda = 4^n$. For sequences o, v with lengths L_o, L_v, write $L_o = N_o T + r_o$ and $L_v = N_v T + r_v$ with $0 \le r_o, r_v < T$. Let S_o (resp. S_v) be the mean block entropy over the N_o (resp. N_v) full T-blocks of o (resp. v). Form $w = o \| v$ and let S_w be the mean over the first $N := N_o + N_v$ full T-blocks of w. Then*

$$S_w = \frac{N_o S_o + N_v S_v}{N}. \tag{20}$$

Moreover, with

$$\theta = \frac{L_o S_o + L_v S_v}{L_o + L_v}, \tag{21}$$

we have the bound

$$\left| \theta - S_w \right| \le \frac{\log(\lambda)}{TN} (r_o + r_v). \tag{22}$$

In particular, when $r_o = r_v = 0$ the identity $\theta = S_w$ holds exactly.

Proof. Write $\theta = \beta S_o + (1-\beta) S_v$ with $\beta = \dfrac{L_o}{L_o + L_v}$ and $S_w = \alpha S_o + (1-\alpha) S_v$ with $\alpha = \dfrac{N_o}{N}$, $N = N_o + N_v$. Then

$$\theta - S_w = (\beta - \alpha)(S_o - S_v), \tag{23}$$

so

$$\left| \theta - S_w \right| \le |\, \beta - \alpha \,| \cdot \left| S_o - S_v \right| \le |\, \beta - \alpha \,| \cdot \log(\lambda). \tag{24}$$

Using $L_o = TN_o + r_o$ and $L_v = TN_v + r_v$ we compute

$$\beta - \alpha = \frac{TN_o + r_o}{TN + r_o + r_v} - \frac{N_o}{N} = \frac{r_o N_v - N_o r_v}{N\big(TN + r_o + r_v\big)}. \tag{25}$$

Hence

$$|\, \beta - \alpha \,| \le \frac{r_o N_v + N_o r_v}{N\big(TN + r_o + r_v\big)} \le \frac{(r_o + r_v)N}{N \cdot TN} = \frac{r_o + r_v}{TN}. \tag{26}$$

Combining the inequalities gives the stated bound. If $r_o = r_v = 0$, then $L_o = TN_o$, $L_v = TN_v$ and

$$\theta = \frac{TN_o S_o + TN_v S_v}{TN} = \frac{N_o S_o + N_v S_v}{N} = S_w. \tag{27}$$

□

We generated three sets of entropy profiles to assess the behavior of the average Shannon entropy S_w under different slicing parameters. In Figures 1 and 2, S_w is shown as a function of the number N of non-overlapping subsequences (blocks) for a fixed sequence length $L = 1000$, using single bases ($n = 1$) and triplets ($n = 3$), respectively. In both cases, the block size was set to

$$T = \left\lfloor \frac{L}{N} \right\rfloor, \tag{28}$$

and any residual bases were discarded. Markers were placed every $\Delta N = 10$.

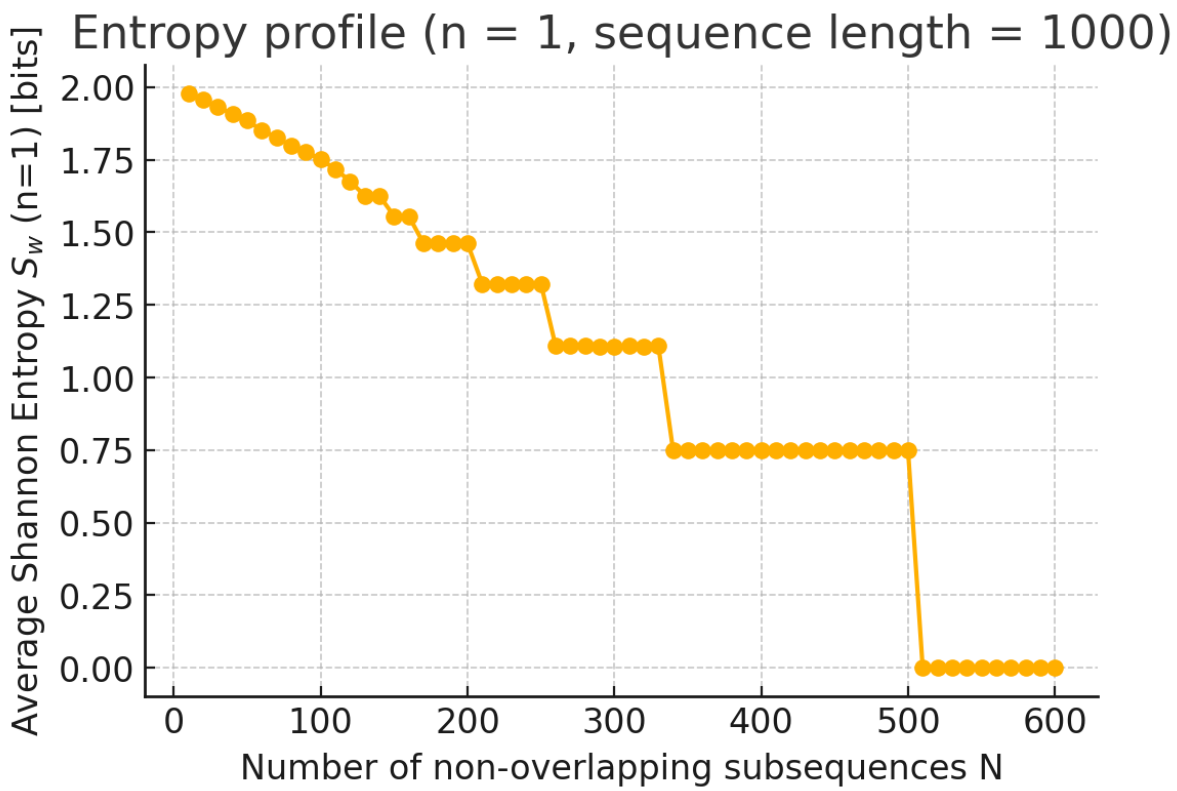

Figure 1: Average entropy S_w of a 1000-base random DNA sequence versus the number of non-overlapping subsequences N, computed with single bases ($n = 1$). Each point is the mean over 50 independent random sequences; the decay is nonlinear and concave, and $S_w \to 0$ once $T = 1$ base.

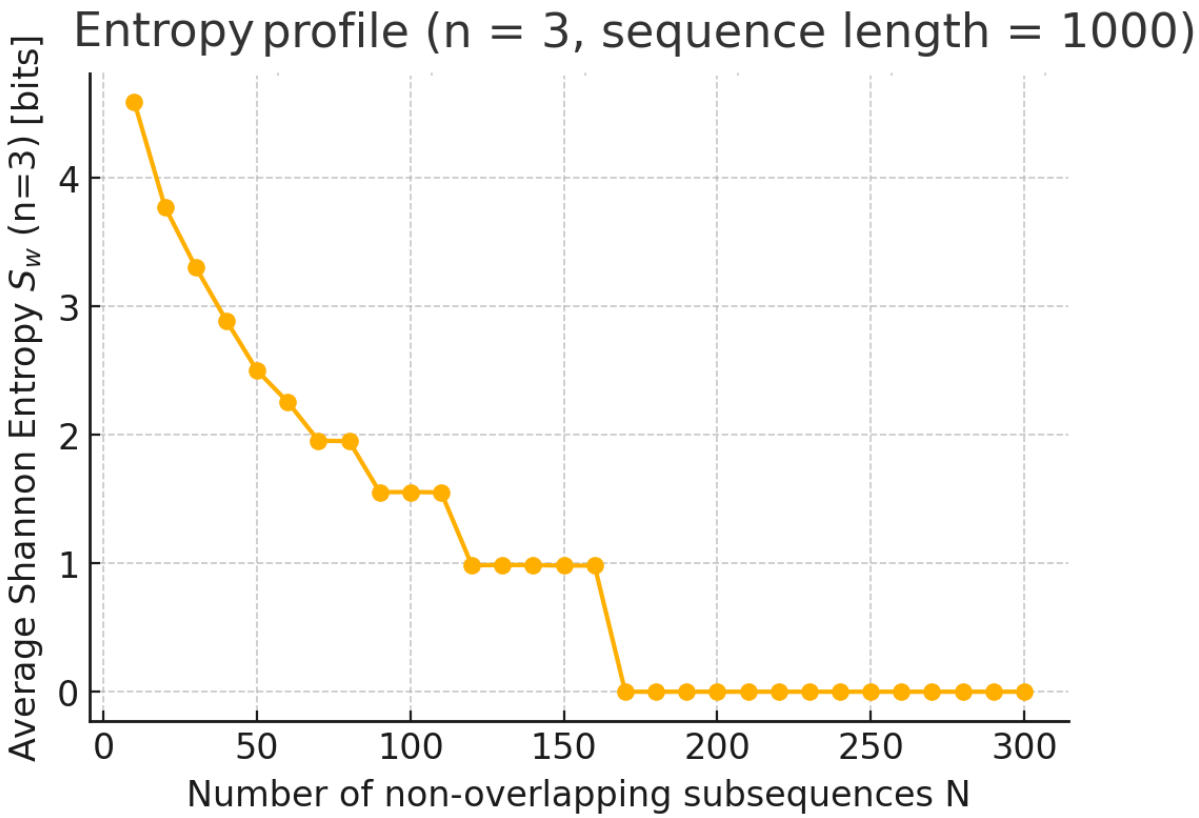

Figure 2: Average entropy S_w for the same 1000-base sequences, using triplets ($n = 3$). Points are averaged over 50 sequences; the entropy falls steeply and reaches zero when blocks no longer contain a full triplet ($T < 3$).

In Figure 3, we fix $N = 1$ (no blocking) and plot the Shannon entropy of the entire sequence as a function of the n-tuple length n, for $1 \leq n \leq 50$. A clear maximum appears when the alphabet size 4^n becomes comparable to the number of observed n-tuples; for $L = 1000$ this happens around $n \approx 6$. Beyond that point, undersampling dominates and entropy decreases as 4^n grows relative to the sample size.

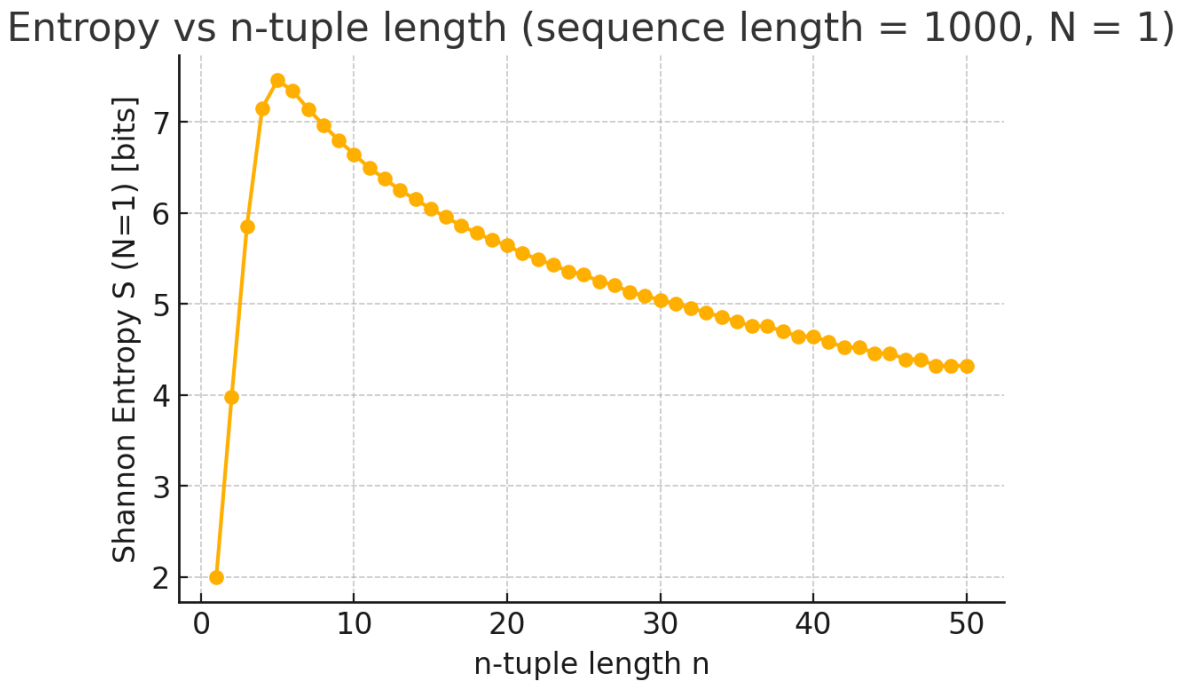

Figure 3: Shannon entropy S of a 1000-base random DNA sequence versus the n-tuple length n (no blocking, $N = 1$), averaged over 50 sequences. The curve peaks near $n \approx 6$ and then falls off as 4^n approaches the sample size.

These profiles were obtained by Monte Carlo simulation over 50 independent random DNA sequences of length 1000. For each sequence and each parameter setting (N or n), the sequence was partitioned into non-overlapping blocks of length $T = \lfloor L/N \rfloor$ (or into n-tuples when $N = 1$), Shannon entropy was computed in each block (or for the full sequence), and then averaged across blocks. The resulting block-average entropies were finally averaged over all 50 sequences to produce the points shown.

It is important to note that the average entropy S_w of a sequence w is *relative* to the chosen values for the n-tuple length n and the subsequence length T. The selection of these parameters must allow the assignment of a meaningful average entropy value. Since the concept of entropy is inherently relative, the numerical value of our average entropy (a real scalar) does not have an absolute interpretation; its meaning depends entirely on the reference system (i.e., the chosen parameters). This is the main limitation of the entropy calculation system based on Shannon entropy (aside from the saturation issues described in Equation (19)).

2.2 Giving Meaning to the Entropy Value

Many previous works have attempted to use alternative entropy systems to obtain distributions or a set of values in order to study the complexity of a sequence[25, 26, 27]. Here, we introduce the concept of the *distribution of all possible sequences of the same length corresponding to a single entropy value.*

To explain the underlying intuition, consider that a DNA sequence w of length L can be one among many sequences of the same length but with different arrangements of letters. For example, take the sequences `AAAG` (denoted o) and `GGTT` (denoted v); both have $L = 4$ but different letter compositions. If we group their letters into frequency vectors with $n = 1$,

$$x_o = (\,3_{\mathrm{A}}, 1_{\mathrm{G}}) \quad \text{and} \quad x_v = (2_{\mathrm{G}},\ 2_{\mathrm{T}}), \tag{29}$$

it is evident that these sequences will, in general, exhibit slightly different entropy values (approximately 0.811278 and 1, respectively). The difference between the entropies of o and v is relative to the parameters chosen; for instance, when $T = 4$ (so that the entire sequence constitutes a single subsequence) and $n = 1$, sequence o yields a lower entropy than v. Such differences are relative and do not possess an absolute meaning when considered in isolation.

Nonetheless, a more meaningful interpretation of the entropy of a sequence can be achieved by relating it not only to another sequence but to *all possible sequences of length* L. In this way, one derives a measure of the intrinsic complexity of a single sequence, independent. To draw an analogy, imagine two individuals with heights of 1.63 meters and 1.89 meters. Without any reference, one may simply say one is shorter than the other, yet absolute labels such as "short" or "tall" only acquire meaning when compared to the average height of a population. Similarly, rather than comparing individual entropy values to a mean, one may count how many DNA subsequences (with fixed length T and non-overlapping n-tuple grouping) exhibit an entropy lower than that of a given sequence w (computed using the same values of T and n). This method yields a distribution of all possible sequences for a given T, ordered by entropy ranging from 0 to $\log(\lambda)$, thus enabling the placement of w within the overall spectrum of complexity.

2.3 Formalizing the Concept

Fix a block (subsequence) length T and an n-tuple size n, with non-overlapping n-tuples inside each block. Let

$$M = \left\lfloor \frac{T}{n} \right\rfloor \tag{30}$$

be the number of n-tuples per block (for $n = 1$ we have $M = T$). The effective alphabet size is $\lambda = \lambda_n = 4^n$. Throughout this section we work at the *block level* (i.e., on a segment of length T).

Definition 2.11. *Let $\aleph_{T,n}$ denote the total number of distinct entropy values obtainable on a block of length T with n-tuples (non-overlapping). Define*

$$Y_{T,n} = \{\, y_1,\ y_2,\ \ldots,\ y_{\aleph_{T,n}} \,\} \tag{31}$$

as the set of these entropy values, ordered so that

$$\log(\lambda) \geq y_{\aleph_{T,n}} \geq \cdots \geq y_2 \geq y_1 = 0. \tag{32}$$

Definition 2.12. *Let*

$$G_{T,n}(y_q) : Y_{T,n} \to \mathbb{N} \tag{33}$$

assign to each $y_q \in Y_{T,n}$ the number of words (blocks) attaining that entropy value.

Definition 2.13. *The total number of possible blocks equals*

$$\sum_{q=1}^{\aleph_{T,n}} G_{T,n}(y_q) \;=\; \lambda^{\,M} \;=\; (4^n)^{\,\lfloor T/n \rfloor}. \tag{34}$$

If $T \bmod n \neq 0$, the last $z = T \bmod n$ symbols of the block are ignored for the entropy computation (truncation); the counting always refers to the truncated word of length nM.

An important insight follows by representing frequency vectors via integer partitions [29]. Let

$$P = (p_1, \ldots, p_\lambda) \tag{35}$$

be an *ordered* partition of the total count c (with $c = T$ for $n = 1$, or $c = M$ for $n > 1$):

$$p_1 \geq \cdots \geq p_\lambda \geq 0, \qquad \sum_{i=1}^{\lambda} p_i = c. \tag{36}$$

Denote by $\mathcal{P}_\lambda(c)$ the set of such ordered partitions.

Definition 2.14. *For $P \in \mathcal{P}_\lambda(c)$ define the Shannon entropy*

$$S(P) = -\sum_{i=1}^{\lambda} \frac{p_i}{c} \log\Big(\frac{p_i}{c}\Big), \tag{37}$$

with the convention $0 \log 0 = 0$. This induces a mapping $S : \mathcal{P}_\lambda(c) \to \mathbb{R}$.

Let $k = \#\{i : p_i > 0\}$ be the number of positive parts. Let $\{\pi_j\}$ be the distinct positive values among $\{p_i\}$ and r_j their multiplicities; set $\gamma = \prod_j r_j!$. The number of words realizing the frequency vector P is

$$O(P) = \binom{\lambda}{k} \frac{k!}{\gamma} \cdot \frac{c!}{\prod_{i=1}^{\lambda} p_i!} = \frac{\lambda!}{(\lambda - k)!\,\gamma} \frac{c!}{\prod_{i=1}^{\lambda} p_i!}. \tag{38}$$

Proposition 2.1 (Sanity check)**.** *We have* $\displaystyle\sum_{P \in \mathcal{P}_\lambda(c)} O(P) = \lambda^c$.

Proof. Let Σ be an alphabet of size λ and fix a length $c \in \mathbb{N}$. For a word $w \in \Sigma^c$ let $f(w) = (f_1, \ldots, f_\lambda) \in \mathbb{N}^\lambda$ be its frequency vector, so that $\sum_{i=1}^{\lambda} f_i = c$. For a fixed frequency vector f, the number of words that realize exactly f is

$$\frac{c!}{\prod_{i=1}^{\lambda} f_i!}. \tag{39}$$

Group frequency vectors f by their *ordered partition* P obtained by sorting the components of f in nonincreasing order. Write $P = (p_1, \ldots, p_\lambda)$ with $p_1 \geq \cdots \geq p_\lambda \geq 0$, $\sum_i p_i = c$, and let $k = \#\{i : p_i > 0\}$. Let $\{\pi_j\}$ be the distinct positive values among $\{p_i\}$ and let r_j be their multiplicities; set $\gamma = \prod_j r_j!$.

For a fixed P, to recover the unsorted frequency vectors f that sort to P: (i) choose which k symbols of Σ are assigned the positive counts ($\binom{\lambda}{k}$ ways); (ii) assign the multiset of positive parts $\{\pi_j\}$ to those k symbols ($k!/\gamma$ ways, accounting for repeated values). Hence the number of frequency vectors f that sort to P is $\binom{\lambda}{k} k!/\gamma = \lambda!/((\lambda - k)!\,\gamma)$. Each such f contributes exactly $c!/\prod_i p_i!$ words (since p_i are the components of f in some order and $0! = 1$). Therefore the total number of words with "shape" P equals

$$O(P) = \frac{\lambda!}{(\lambda - k)!\,\gamma} \frac{c!}{\prod_{i=1}^{\lambda} p_i!}. \tag{40}$$

Summing over all ordered partitions $P \in \mathcal{P}_\lambda(c)$ thus yields

$$\sum_{P \in \mathcal{P}_\lambda(c)} O(P) = \sum_{\substack{f_1, \ldots, f_\lambda \geq 0 \\ f_1 + \cdots + f_\lambda = c}} \frac{c!}{\prod_{i=1}^{\lambda} f_i!} = (1 + \cdots + 1)^c = \lambda^c. \tag{41}$$

The last equality is the multinomial theorem applied to $(x_1 + \cdots + x_\lambda)^c$ with $x_i \equiv 1$. $\square$

Since different partitions can share the same entropy, define the discrete distribution

$$G_{T,n}(y) \;=\; \sum_{P\,:\,S(P)=y} O(P). \tag{42}$$

This is the distribution used in Definitions 2.11–2.13.

We can now construct $G_{T,n}(y)$. For example, Figure 4 shows the case $T = 20$ and $n = 1$, where $c = T$ and $\lambda = 4$.

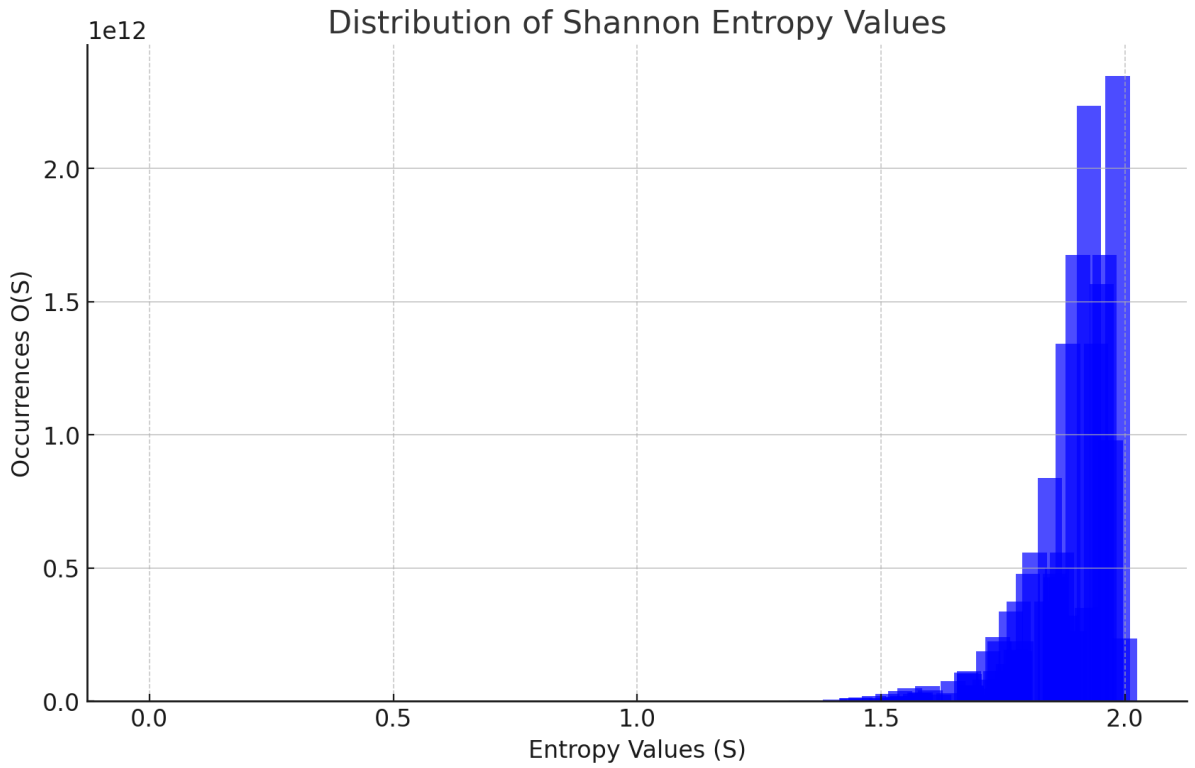

Figure 4: Distribution G of occurrences O as a function of entropy values S (plotted on the horizontal axis, arranged in increasing order from 0 to 2.0), using $T = 20$ and $n = 1$. As described earlier, G is a discrete distribution.

Definition 2.15 (Entropy–rank ratio)**.** *Fix the subsequence length T, the n-tuple size n, and the number of blocks N. Let $S^{(N)}_{T,n}(w)$ be the mean Shannon entropy of w computed by splitting w into N consecutive blocks of length T and averaging their block entropies (with non-overlapping n-tuples inside each block). Let $G^{(N)}_{T,n}$ be the distribution of the mean of N i.i.d. block entropies, i.e. the N-fold discrete convolution of $G_{T,n}$ for the sum, followed by a scaling of the support by $1/N$.*

The entropy–rank ratio *of w is*

$$R^{(N)}_{T,n}(w) \;=\; \frac{\displaystyle\sum_{y \le S^{(N)}_{T,n}(w)} G^{(N)}_{T,n}(y)}{\displaystyle\sum_{y} G^{(N)}_{T,n}(y)} \;\in (0,1]. \tag{43}$$

Practical note. In our implementation we use $N = 1$, so $S^{(1)}_{T,n}(w)$ is the entropy of a single block of length T and $G^{(1)}_{T,n} \equiv G_{T,n}$.

Proposition 2.2 (Block-mean counting as N-fold convolution)**.** *Fix T, n and let $G_{T,n}(y)$ be the block-level counting measure of entropies on length-T blocks. Consider sequences of length NT segmented in N consecutive T-blocks. Let $G_{T,n}^{(N)}(s)$ count how many such sequences have mean block entropy exactly $s \in \frac{1}{N}\sum_{i=1}^{N} \mathrm{supp}(G_{T,n})$. Then*

$$G_{T,n}^{(N)}(s) = \sum_{\substack{y_1,\ldots,y_N \\ (y_1+\cdots+y_N)/N=s}} \prod_{i=1}^{N} G_{T,n}(y_i), \tag{44}$$

i.e. $G_{T,n}^{(N)}$ is the N-fold discrete convolution of $G_{T,n}$ for the sum, followed by a scaling of the support by $1/N$. Moreover $\sum_s G_{T,n}^{(N)}(s) = \big(\sum_y G_{T,n}(y)\big)^N$.

Proof. Every length-NT sequence is an ordered concatenation of N length-T blocks. The number of concatenations realizing a fixed N-tuple $(y_1, \ldots, y_N)$ is $\prod_i G_{T,n}(y_i)$. Grouping by the mean $s = (y_1 + \cdots + y_N)/N$ yields the formula. The total count follows by the multiplicative principle. □

Proposition 2.3 (Calibration of R)**.** *Let Y be a random entropy drawn from the counting measure $G_{T,n}$ on length-T blocks (uniform over all blocks after truncation). Define the right-continuous cdf*

$$F(y) = \frac{\sum_{u \le y} G_{T,n}(u)}{\sum_u G_{T,n}(u)}. \tag{45}$$

Then $R = F(Y)$ is super-uniform: for all $t \in [0, 1]$,

$$\mathbb{P}(R \le t) \;\le\; t, \tag{46}$$

with equality if ties at atoms of F are broken uniformly at random. The same statement holds for N blocks with $G_{T,n}^{(N)}$.

Proof. This is the standard property of the cdf in the discrete case: $F(Y)$ is right-continuous uniform (or super-uniform without randomized tie-breaking). The N-block case follows by replacing $G_{T,n}$ with $G_{T,n}^{(N)}$. □

2.4 Implementation

In our implementation, the DNA sequences are first converted into integer arrays by mapping each nucleotide A, C, G, T to 0, 1, 2, 3, respectively for tokenization, while reserving the value 4 for the padding symbol. For the purpose of entropy computation, each sequence is partitioned into non-overlapping n-tuples and the frequency vector is computed from these groups. The Shannon entropy is then calculated based on Equation (17). Data augmentation is achieved by cropping the sequence into fixed-length segments using either:

1. A Random Crop.
2. A Kolmogorov complexity-based crop, that selects subsequences whose Kolmogorov complexity, calculated with the help of the zlib library, is close to that of the entire sequence.
3. An entropy-based crop that selects subsequences whose Shannon entropy is close to that of the entire sequence.
4. A Ratio-based crop that selects subsequences whose entropy ratio R is close to that of the entire sequence.
5. Basic Crop that provides technically no real augmentation which will be the reference

(we are not comparing against full-sequence models, as our sole objective is to evaluate the efficiency of the complexity measures).
The CNN classifier uses an embedding layer with a vocabulary size of 5 (corresponding to the four nucleotides plus the padding token), followed by convolutional layers, adaptive pooling, dropout, and a fully connected layer for classification. Although many CNN-based approaches utilize k-mer tokenization[33] with larger vocabularies, our implementation adheres to single-nucleotide tokenization combined with non-overlapping n-tuple based entropy analysis.

Algorithm 1 Random Crop

procedure RANDOMCROP(seq, target_len, offset_ratio)
  input: seq, target_len, offset_ratio
  if length(seq) $\leq$ target_len **then**
    return `pad_or_sample_sequence`(seq, target_len)
  else
    center $\leftarrow \left\lfloor \dfrac{\text{length(seq)} - \text{target\_len}}{2} \right\rfloor$
    max_offset $\leftarrow \lfloor (\text{length(seq)} - \text{target\_len}) \times \text{offset\_ratio} \rfloor$
    offset $\leftarrow$ RandomInteger($-$max_offset, max_offset)
    start $\leftarrow$ Clip(center + offset, 0, length(seq) $-$ target_len)
  endif
  return seq[start : start + target_len]
end procedure

Algorithm: Random Crop.

This procedure crops a sequence to a fixed length while allowing controlled randomness around its center. If the sequence is shorter than or equal to the target length, it is processed by `pad_or_sample_sequence`. Otherwise, the center of the sequence is computed, a maximum offset is determined by the given `offset_ratio`, and a random offset is applied. The starting index is then clipped to ensure it is within the valid range, and the fixed-length subsequence is returned.

Algorithm 2 R-based Crop

procedure RatioBasedCrop(seq, target_len, num_candidates, offset_ratio, ratio_whole_seq, T, n, α, β)
  input: seq, target_len, num_candidates, offset_ratio, ratio_whole_seq, T, n, α, β
  if length(seq) $\leq$ target_len **then**
    return `pad_or_sample_sequence`(seq, target_len)
  endif
  arr $\leftarrow$ `Array`(seq) ▷ 0-based indexing
  $L \leftarrow$ length(arr)
  center $\leftarrow \left\lfloor \dfrac{L - \text{target\_len}}{2} \right\rfloor$
  max_offset $\leftarrow \lfloor (L - \text{target\_len}) \times \text{offset\_ratio} \rfloor$
  candidate_offsets $\leftarrow$ `RandomIntegers`($-$max_offset, max_offset, num_candidates)
  best_score $\leftarrow \infty$; best_candidate $\leftarrow$ None
  for $i \leftarrow 0$ **to** num_candidates $- 1$ **do**
    start $\leftarrow$ center + candidate_offsets[i]
    if start < 0 **then** start $\leftarrow 0$ **endif**
    if start $> L -$ target_len **then** start $\leftarrow L -$ target_len **endif**
    segment $\leftarrow$ arr[start : start + target_len]
    cand_ratio $\leftarrow$ `calculate_ratio`(segment, T, n) ▷ same (T, n) as `ratio_whole_seq`
    ratio_diff $\leftarrow |\,\text{cand\_ratio} - \text{ratio\_whole\_seq}\,|$
    if max_offset > 0 **then**
      offset_penalty $\leftarrow \dfrac{|\,\text{candidate\_offsets}[i]\,|}{\text{max\_offset}}$
    else
      offset_penalty $\leftarrow 0$
    endif
    score $\leftarrow \alpha \times$ ratio_diff $+ \beta \times$ offset_penalty
    if score $<$ best_score **then**
      best_score $\leftarrow$ score; best_candidate $\leftarrow$ segment
    endif
  endfor
  return best_candidate
end procedure

Algorithm: R-based Crop.
This procedure crops a sequence to a fixed length using a ratio-based selection mechanism. If the sequence is shorter than or equal to the target length, it is processed by `pad_or_sample_sequence`. Otherwise, the sequence is converted into an array, a central index is computed, and a set of candidate offsets is generated. For each candidate, the starting index is clipped, and a subsequence is extracted. A ratio for the candidate segment is computed and compared to the provided `ratio_whole_seq`. A score is calculated by combining the ratio difference (weighted by α) and an offset penalty (weighted by β). The candidate with the minimum score is selected and returned as the best representative subsequence.

Note: Parameters α and β weigh, respectively, ratio coherence and offset penalty; they must be tuned on validation (e.g., grid/random search). Setting $\beta = 0$ removes recentering pressure.

Algorithm 3 Compress Subchunk

```
procedure compressSubchunk(chunk)
    if size(chunk) = 0 then
        return (0, chunk)
    endif
    bytes_buf ← PackAsBytes(chunk)                ▷ e.g. cast to uint8 and pack values 0–4
    cached_val ← Retrieve bytes_buf from _KOLMOGOROV_CACHE
    if cached_val is not None then
        return (cached_val, chunk)
    endif
    comp ← Compress(bytes_buf, level = 1)
    length ← Length(comp)
    if Size(_KOLMOGOROV_CACHE) < _KOLMOGOROV_CACHE_MAX_SIZE then
        Store (bytes_buf, length) in _KOLMOGOROV_CACHE
    endif
    return (length, chunk)
end procedure
```

Algorithm: Compress Subchunk.
This procedure compresses a given subchunk. If the subchunk is empty, it returns zero. Otherwise, it packs the integers into a binary buffer (e.g., uint8), checks a cache for a precomputed compressed length, compresses the buffer if needed, caches the result (subject to a maximum size), and returns the compressed length along with the original subchunk.

Algorithm 4 Kolmogorov-based Crop

```
procedure KOLMOGOROVBASEDCROP(arr, target_len, offset_ratio = OFFSET_RATIO, pick = 'max', num_candidates = NUM_CANDIDATES)
    L ← length(arr)
    if L ≤ target_len then
        return arr
    endif
    center ← (L − target_len)//2
    max_offset ← ⌊(L − target_len) × offset_ratio⌋
    tasks ← empty list
    for j ← 1 to num_candidates do
        off ← randint(−max_offset, max_offset)
        start ← center + off
        if start < 0 then
            start ← 0
        elseif start > L − target_len then
            start ← L − target_len
        endif
        chunk ← arr[start : start + target_len]
        Append chunk to tasks
    endfor
    if pick = 'max' then
        best_val ← −∞
    else
        best_val ← ∞
    endif
    best_chunk ← None
    with ThreadPoolExecutor(max_workers = NUM_KOLMOGOROV_THREADS) as ex
        for each (comp_len, chunk) in ex.map(COMPRESSSUBCHUNK, tasks)
            if (pick = 'max' and comp_len > best_val) or
                    (pick ≠ 'max' and comp_len < best_val) then
                best_val ← comp_len
                best_chunk ← chunk
            endif
    return best_chunk if best_chunk ≠ None else arr
end procedure
```

Algorithm: Kolmogorov-based Crop.
This procedure crops an array to a fixed length using a Kolmogorov complexity–based criterion. If the array length is less than or equal to the target length, it returns the original array. Otherwise, it generates multiple candidate subarrays by applying random offsets around the center. For each candidate, it computes a compressed length using the COMPRESSSUBCHUNK procedure. Depending on whether the selection criterion is 'max' or otherwise, it selects the candidate with the maximum or minimum compressed length and returns it.

Algorithm 5 Entropy-based Crop

procedure EntropyBasedCrop(arr, target_len, full_entropy, n, num_candidates, offset_ratio, α, β)
  $L \leftarrow$ length(arr)
  if $L \leq$ target_len **then**
    return arr
  endif
  center $\leftarrow \left\lfloor \dfrac{L - \text{target\_len}}{2} \right\rfloor$
  max_offset $\leftarrow \lfloor (L - \text{target\_len}) \times \text{offset\_ratio} \rfloor$
  allocate arrays offsets[0..num_candidates − 1], scores[0..num_candidates − 1], starts[0..num_candidates − 1]
  for $i = 0$ **to** num_candidates − 1 **do**
    offsets[i] $\leftarrow$ RandomInteger(−max_offset, max_offset)
  endfor
  for $i = 0$ **to** num_candidates − 1 **do**
    off $\leftarrow$ offsets[i]
    start $\leftarrow$ Clip(center + off, 0, L − target_len)
    starts[i] $\leftarrow$ start
    chunk $\leftarrow$ arr[start : start + target_len]
    cand_ent $\leftarrow$ `compute_n_gram_entropy_jit`(chunk, n) ▷ entropy of the chunk, $N = 1$
    ent_diff $\leftarrow \left|\, \text{cand\_ent} - \text{full\_entropy} \,\right|$
    if max_offset > 0 **then**
      offset_penalty $\leftarrow |\text{off}|/\text{max\_offset}$
    else
      offset_penalty $\leftarrow 0$
    endif
    scores[i] $\leftarrow \alpha \times$ ent_diff $+ \beta \times$ offset_penalty
  endfor
  best_idx $\leftarrow \arg\min_i$ scores[i]
  return arr[starts[best_idx] : starts[best_idx] + target_len]
end procedure

Algorithm: Entropy-based Crop.
This procedure crops an array to a fixed length by selecting the segment whose n-gram entropy best approximates a given full entropy value. If the array is too short, it returns the original array. Otherwise, it generates multiple candidate segments using random offsets around the center. For each candidate, it computes the entropy difference and applies an offset penalty. The candidate with the minimal combined score (weighted by α and β, two real numbers) is chosen and returned. Any n-gram containing the padding token is ignored in the entropy calculation.

All the previous algorithms are precomputed augmentations that are needed to achieve higher precisions and performances. It is our goal to demonstrate whether Shannon entropy and Ratio are useful in order to improve previous algorithms.

Algorithm 6 Algorithm: CNN Classifier Initialization

```
procedure INIT(vocab_size, token_dim, num_classes)
    input: vocab_size, token_dim, num_classes
    embedding ← EmbeddingLayer(vocab_size, token_dim, padding_idx=4)
    conv1 ← Conv1D(in_channels=token_dim, out_channels=16, kernel_size=3, stride=1, padding=1)
    conv2 ← Conv1D(in_channels=16, out_channels=16, kernel_size=3, stride=1, padding=1)
    pool ← AdaptiveMaxPool1D(1)
    dropout ← Dropout(MODEL_DROPOUT)
    relu ← ReLU()
    input_dim ← 16
    fc_final ← FullyConnected(input_dim, num_classes)
end procedure
```

Algorithm: CNN Classifier Initialization.
This algorithm initializes the CNN classifier by setting up the embedding layer, convolutional layers, pooling and dropout.

Algorithm 7 CNN Forward Pass

```
procedure FORWARDPASS(input_sequence)
    x ← EmbeddingLayer(input_sequence)
    x ← Transpose(x)                          ▷ Adjust dimensions for convolution
    x ← Convolutional layers with ReLU activations
    x ← AdaptiveMaxPool1D(x)
    x ← Dropout(x)
    return FullyConnected(x)
end procedure
```

Algorithm: CNN Forward Pass.
This algorithm embeds the raw viral single nucleotide-DNA tokens into vector form (or trinucleotide-DNA tokens for mammal genes), transposes the resulting tensor to match the expected convolutional input format, applies convolutional filters with ReLU activations to extract local sequence patterns, reduces dimensionality via adaptive pooling, and finally returns classification logits through a fully connected layer.

2.5 Comparisons between the Models

In our implementation, DNA sequences are not tokenized into trinucleotides, in order to deploy even less memory. Instead, each nucleotide is mapped to an integer in $\{0, 1, 2, 3, 4\}$ (corresponding to A, C, G, T and the padding symbol). For the CNN, the vocabulary size is therefore 5. Two configurations have been considered: one with a token dimension of 8 and one with a token dimension of 256. The parameter counts are computed as follows.

For the configuration with token dimension 8, the embedding layer has $5 \times 8 = 40$ parameters. The first convolutional layer has $8 \times 16 \times 3 + 16 = 400$ parameters. The second convolutional layer has $16 \times 16 \times 3 + 16 = 784$ parameters. The final fully connected layer has $16 \times 6 + 6 = 102$ parameters. The total is $40 + 400 + 784 + 102 = 1326$ parameters.

For the configuration with token dimension 256, the embedding layer has $5 \times 256 = 1280$ parameters. The first convolutional layer has $256 \times 16 \times 3 + 16 = 12304$ parameters. The second convolutional layer has $16 \times 16 \times 3 + 16 = 784$ parameters. The final fully connected layer has $16 \times 6 + 6 = 102$ parameters. The total is $1280 + 12304 + 784 + 102 = 14470$ parameters.

Configuration	Parameter Calculation	Total
Token Dim = 8	Embedding: $5 \times 8 = 40$ CNN Layers: Conv1: $8 \times 16 \times 3 + 16 = 400$ Conv2: $16 \times 16 \times 3 + 16 = 784$ FC Final: $16 \times 6 + 6 = 102$ Total: $40 + 400 + 784 + 102 = 1326$	1326
Token Dim = 256	Embedding: $5 \times 256 = 1280$ CNN Layers: Conv1: $256 \times 16 \times 3 + 16 = 12304$ Conv2: $16 \times 16 \times 3 + 16 = 784$ FC Final: $16 \times 6 + 6 = 102$ Total: $1280 + 12304 + 784 + 102 = 14470$	14470

Table 1: Comparison of the number of parameters in the CNNs. In our implementation, the DNA sequence is represented using a vocabulary of 5 (one per nucleotide plus padding), with two configurations considered: one with token dimension 8 and one with token dimension 256.

3 Results

3.1 Viral Genes

Two series of tests were conducted using the benchmark code with the implementation of several CNNs using the dataset of viral genes from [14]. A total of **five** CNNs were implemented based on the previously described algorithms: for both the configuration with token dimension 8 and that with token dimension 256, we implemented one model using data augmentation via Random Crop, one using data augmentation via R-Based Crop, one using data augmentation via Entropy-based Crop, one using data augmentation via Kolmogorov Complexity algorithm, one without data augmentation. For the token-dim = 256 model, we only used 28 sequences for training and 13 for initial validation, then tested on 400 other sequences 40 times to get meaningful average test accuracies.[1] For the token-dim = 8 model we used the whole viral training set (1320 sequences). Here are the results of the benchmarks:

Table 2: CNN results (Token Dim = 256, training sequences = 28, development sequences = 13, T = 22, n = 1; macro-averaged metrics over 40 iterations) on `test_set.csv` (400 sequences)

Variant	Accuracy (mean ± std)	Recall (mean ± std)	F1 (mean ± std)
Random Crop	0.242 ± 0.034	0.265 ± 0.037	0.226 ± 0.039
Ratio-based Crop	0.859 ± 0.048	0.875 ± 0.044	0.865 ± 0.045
Entropy-based Crop	0.747 ± 0.069	0.770 ± 0.060	0.752 ± 0.065
Kolmogorov-based Crop	0.393 ± 0.042	0.410 ± 0.042	0.380 ± 0.041
No augmentation (basic crop)	0.236 ± 0.033	0.256 ± 0.032	0.219 ± 0.034

Unless otherwise stated, the 95% margin of error (MOE) is computed as

$$\text{MOE} \;=\; 1.96\,\frac{\sigma}{\sqrt{R}}, \tag{47}$$

[1] For each of the 40 runs, training was re-initialised from scratch with *different* random seeds, ensuring that the model never saw the test set during optimisation.

where σ is the standard deviation across the $R = 40$ random restarts.

Table 3: CNN results (Token Dim = 8, training sequences = 1320, development sequences = 180, T = 22, n = 1; macro-averaged metrics over 40 iterations) on `test_set.csv` (400 sequences)

Variant	**Accuracy** (mean ± std)	**Recall** (mean ± std)	**F1** (mean ± std)
Entropy-based Crop	0.893 ± 0.036	0.899 ± 0.032	0.893 ± 0.035
Ratio-based Crop	0.926 ± 0.026	0.928 ± 0.025	0.924 ± 0.026
Random Crop	0.468 ± 0.040	0.481 ± 0.043	0.461 ± 0.040
Kolmogorov-based Crop	0.625 ± 0.029	0.640 ± 0.029	0.620 ± 0.029
No augmentation (basic crop)	0.466 ± 0.034	0.480 ± 0.035	0.458 ± 0.033

The 95% confidence-interval margin of error for accuracy is approximately ±1.1% for the entropy-based method, ±0.8% for ratio-based, and around ±1% for the remaining variants.

3.2 Human Genes with Repetitions

For this other dataset [15] we only used the token-dim = 256 configuration because of the scarcity of the data: only 11 sequences associated with expansion pathologies, while 13 not associated. A total of 24 sequences, split into 16 for training and 8 for testing over 40 iterations to get sound average accuracies.[2] Here we implemented the same cropping techniques as in the previous set of tests:

Table 4: CNN results (Token Dim = 256, training sequences = 16, test sequences = 8, T = 98, n = 2; macro-averaged metrics over 40 iterations)

Variant	**Accuracy** (mean ± std)	**Recall** (mean ± std)	**F1** (mean ± std)
Random Crop	0.559 ± 0.068	0.559 ± 0.068	0.535 ± 0.073
Ratio-based Crop	0.741 ± 0.047	0.741 ± 0.047	0.731 ± 0.051
Entropy-based Crop	0.666 ± 0.046	0.666 ± 0.046	0.653 ± 0.048
Kolmogorov-based Crop	0.569 ± 0.050	0.569 ± 0.050	0.551 ± 0.053
No augmentation (basic crop)	0.584 ± 0.050	0.584 ± 0.050	0.564 ± 0.053

The 95% margin of error for the accuracy estimates ranges from approximately ±2.1 % for Random Crop to ±1.5 % Ratio-based Crop and ±1.4% for Entropy-based Crop, with intermediate values of ±1.6% for both Kolmogorov-based Crop and No augmentation.

4 Discussion

4.1 Viral Genes

Our tests have highlighted that data augmentation techniques based on the introduction of the parameter R, which guides the data augmentation, **consistently** improve performance. The accuracy achieved with local pattern extractions neural network models such as CNNs on the viral DNA dataset, trained on only 41 sequences, is remarkably high (86.1%), suggesting important potential applications in the study of cases where the number of available samples is extremely limited.

Furthermore, the accuracy obtained (92.4%) by the model with only 1326 parameters, occupying less than 10 kilobytes in `FP32` format, suggests practical implementations of

[2]Each of the 40 runs started from randomly initialised weights and did not reuse the test sequences during training.

very small neural networks capable of performing inference on devices with minimal RAM (only a few megabytes). This opens the door to the development of compact genetic self-testing devices.

These results would not be possible without the entropy distribution calculation discussed earlier and the formal introduction of the parameter R. In fact, the other data augmentation techniques are clearly not as precise as our novel method based on R (although it is to be noted that the n-gram entropy cropping is novel too). This also suggests profound biological and genetic implications, where our entropy, given its effectiveness in distinguishing different gene types, as demonstrated in the benchmark, plays a fundamental role in decoding the genetic information contained in DNA.

We can explain these consistent improvements in performance by analysing the distribution of sequences from the viral gene dataset (*training_set.csv*) on a plane. We define a plane where the vertical axis represents the GC content (commonly used in these classification problems) and the horizontal axis represents:

1. The Kolmogorov Complexity calculated on zlib compression.

2. The classical Shannon Entropy on the full sequences with $n = 1$.

3. The Ratio of the sequences with $T = 22, n = 1$.

We obtain the following distributions for the six classes:

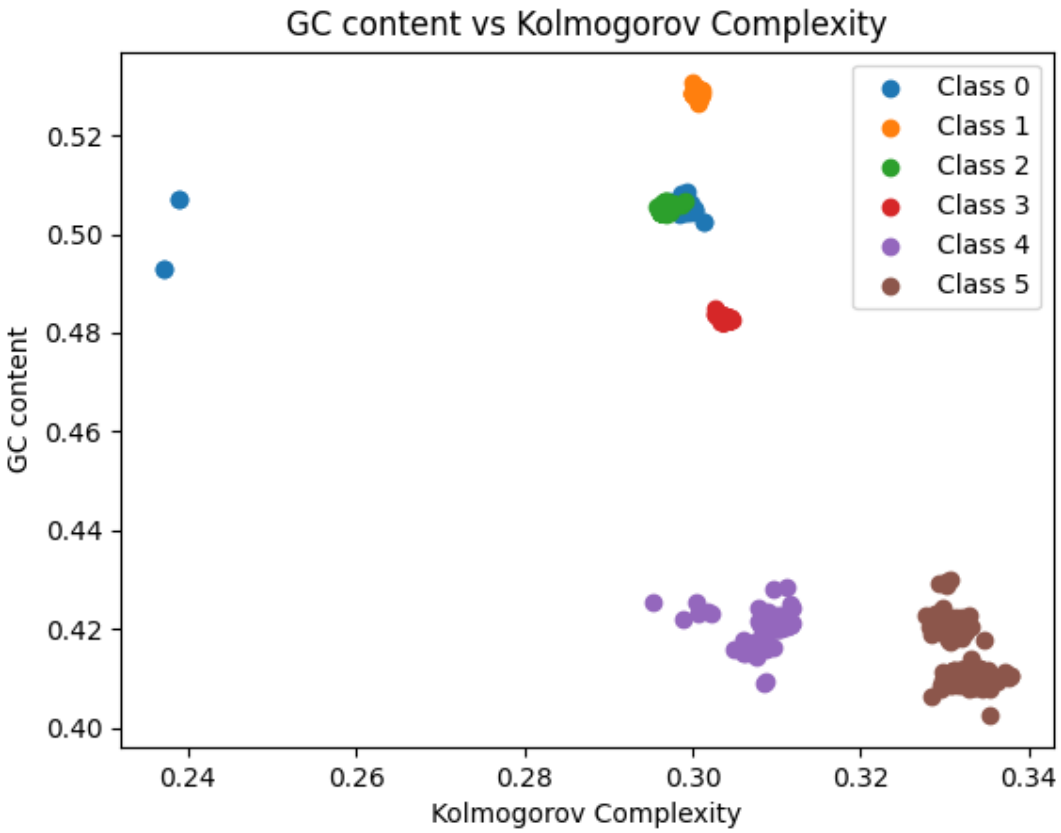

Figure 5: Distribution of the six viral classes *training_set.csv* on the GC content–Kolmogorov Complexity (based on zlib compression) plane.

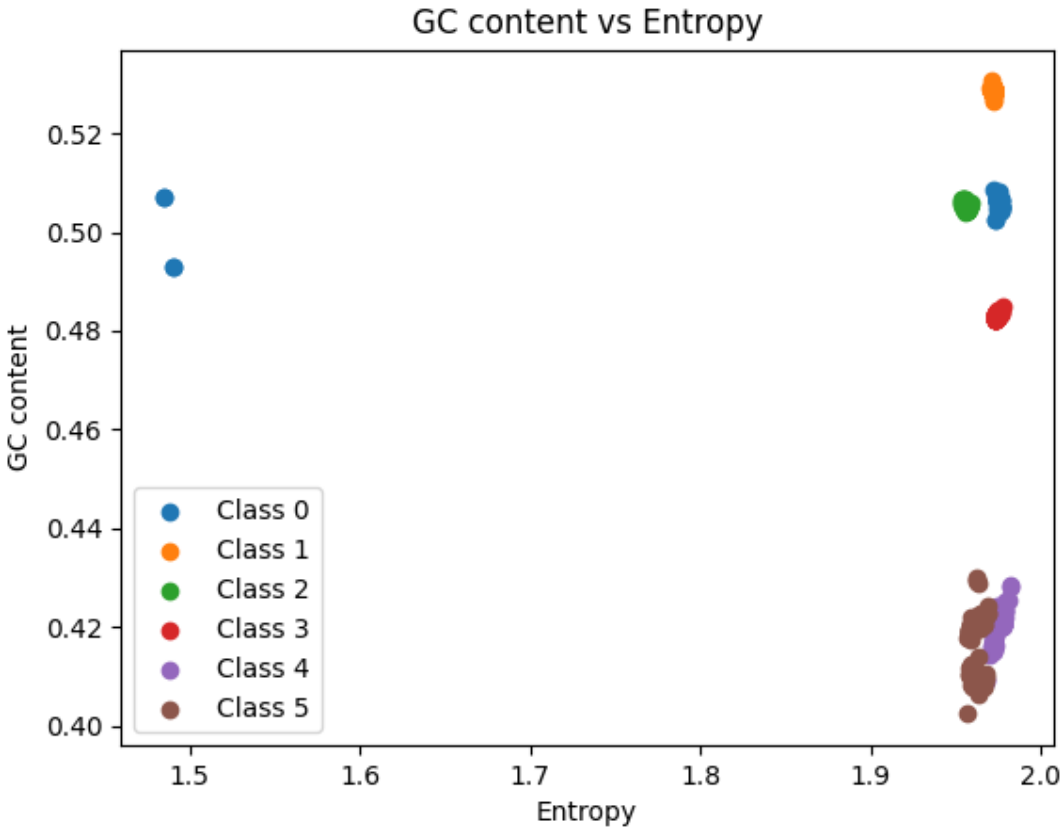

Figure 6: Distribution of the six viral classes *training_set.csv* on the GC content–Shannon Entropy $n = 1$ plane.

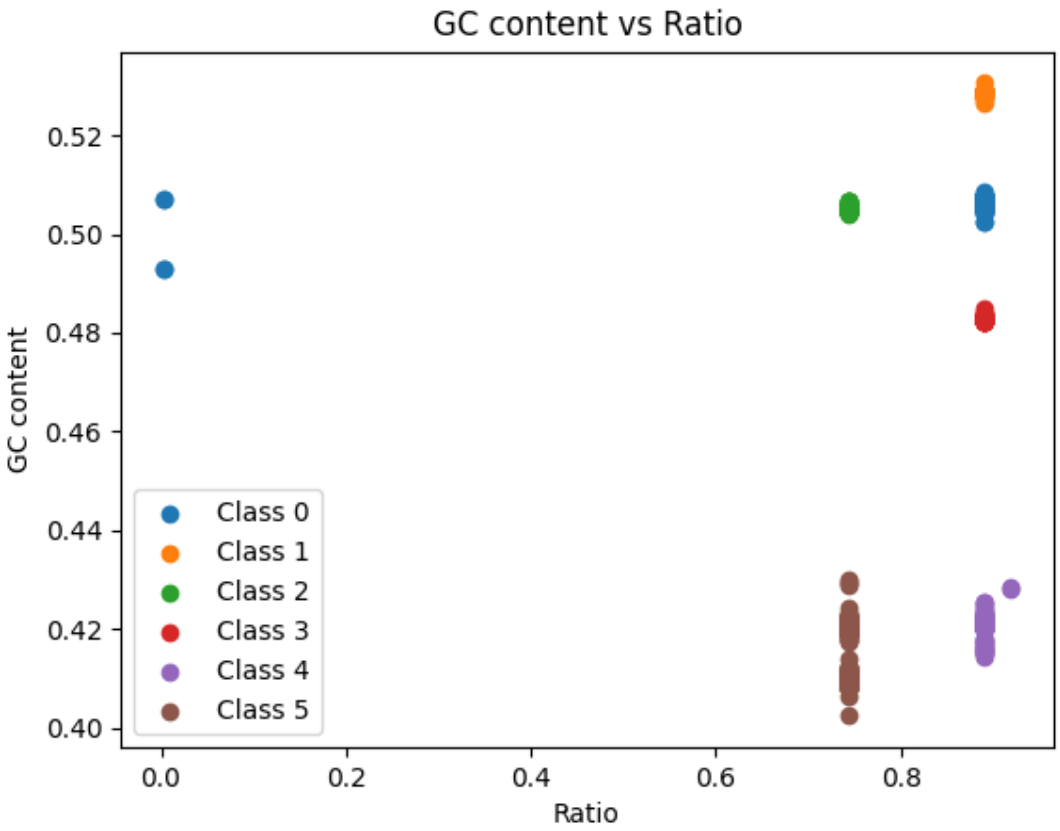

Figure 7: Distribution of the six viral classes *training_set.csv* on the GC content–Ratio with $T = 22, n = 1$ plane.

As we can see from the distributions, we have overlapping classes for the Kolmogorov Complexity (class 0 and class 2) and the Shannon Entropy (class 4 and class 5). For the R-based distribution of the DNA sequences, we have almost perfect discrimination among sequences with different R value and same GC Content value. Furthermore, the difference between classes with similar amount of GC content is larger in the case of Ratio (tenths), while the other methods only differentiate classes in the order of cents.

It is logical to think that cropping algorithms based on R will not only be more precise, but easier to fine-tune, being more stable on changes in the parameters of the Neural Network.

4.2 Human Genes with Repetitions

The value of these results is further supported by the sound results obtained on the dataset of human DNA sequences subject to expansion (73.4% with only 24 sequences using the R-based crop).

We can plot similar distributions over this dataset, with similar results to the previous ones, using $n = 2$ for Shannon Entropy and $T = 98, n = 2$ for Ratio:

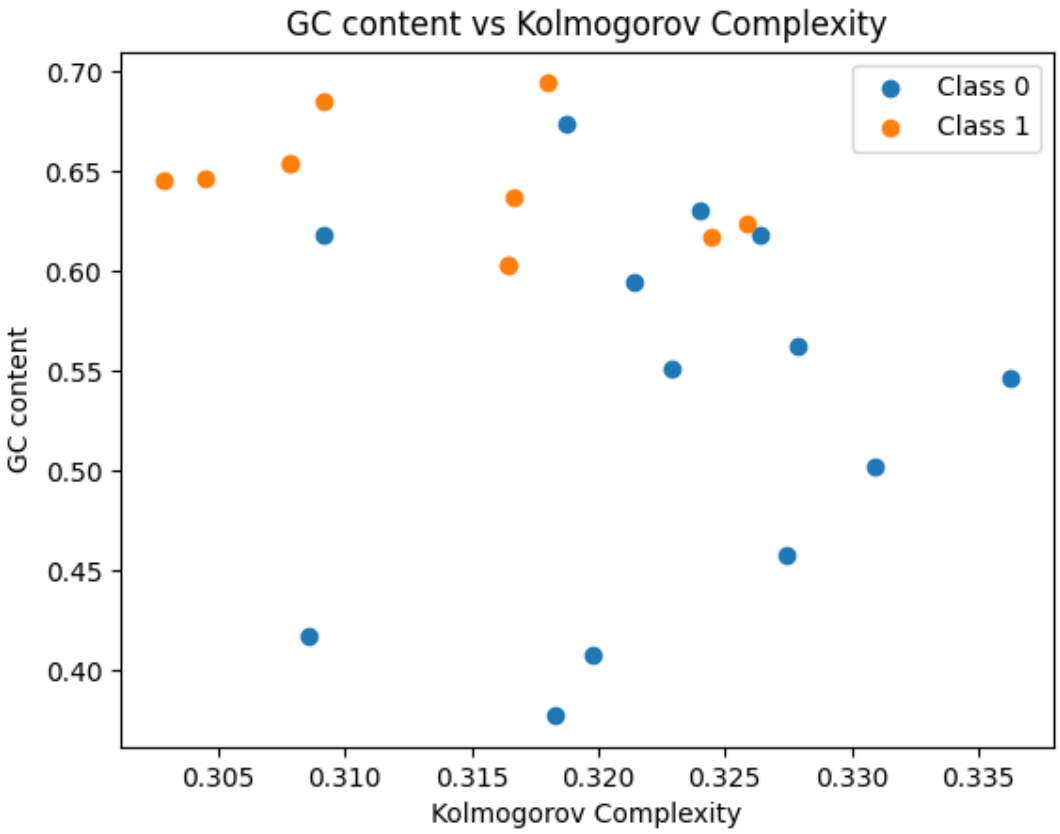

Figure 8: Distribution of the two classes (pathological and not pathological) *human_training_set.csv* on the GC content–Kolmogorov Complexity plane.

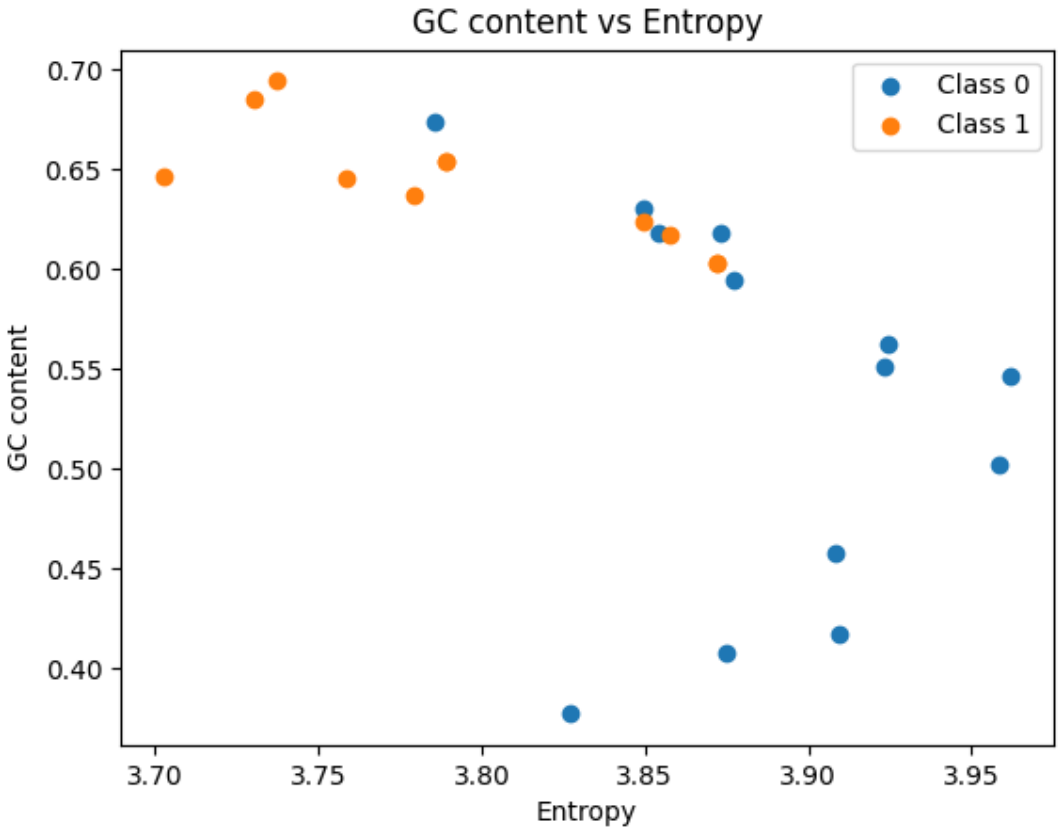

Figure 9: Distribution of the two classes (pathological and not pathological) *human_training_set.csv* on the GC content– Shannon Entropy $n = 2$ plane.

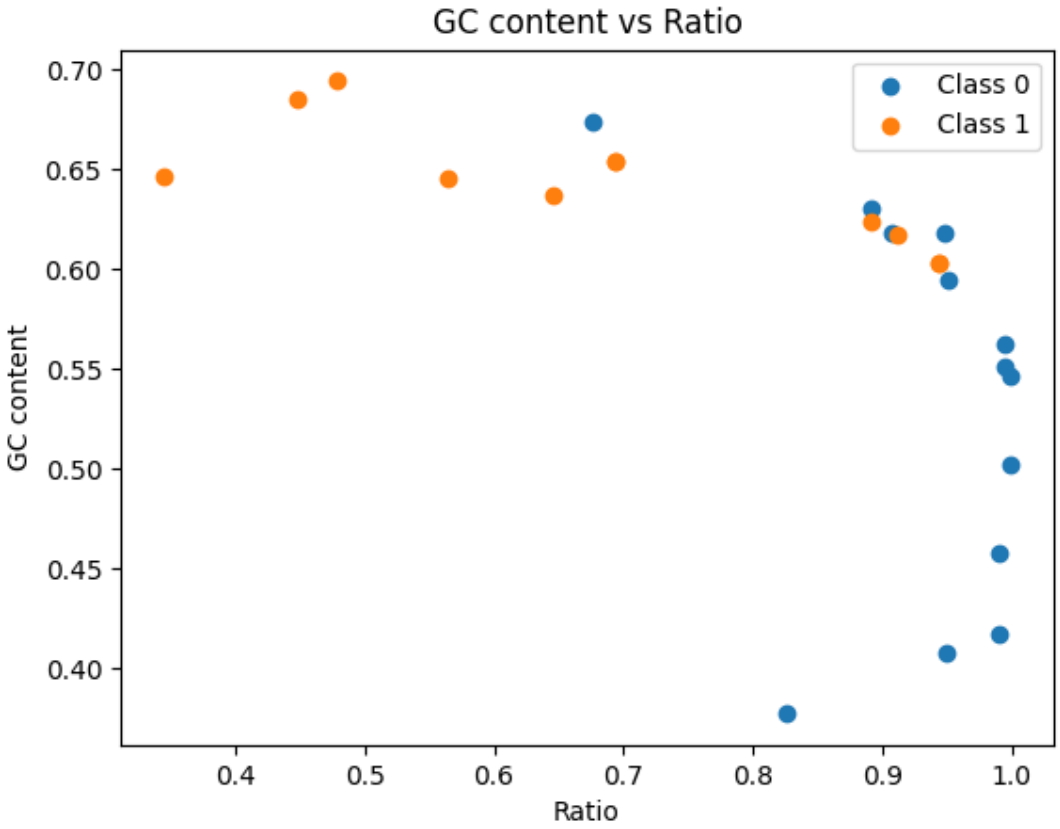

Figure 10: Distribution of the two classes (pathological and not pathological) *human_training_set.csv* on the GC content–Ratio $T = 98, n = 2$ plane.

As we can see, while Kolmogorov Complexity gives less precise distinction between the two classes, both Shannon Entropy and Ratio give similar results, but the parameter R provides again a larger more precise representation and a larger gap between the two main clusters.

Hence, we can understand that, often, R tends to form a distinct cluster (with only negligible exceptions). This greatly facilitates the task of classifying DNA sequences.

4.2.1 On information content.

At fixed (T, n), R is a monotone transform of the block entropy and therefore does not add raw information beyond S; its benefit is that it normalizes values with respect to the full combinatorial distribution, making them directly comparable across sequences.

5 Conclusions

Our work shows that it is possible to introduce a new perspective on Shannon Entropy for DNA, based on the novel parameter R, defined as the ratio that locates a target sequence within the distribution of all possible sequences of the same length. This is achieved by subdividing the sequence into fixed-length subsequences and non-overlapping n-tuples. By doing so, and thereby avoiding the limitations of classical Shannon Entropy (such as the tendency of entropy values to approach 2 for most sequences) we developed cropping algorithms based on the parameter R that **consistently** increase the precision

of CNNs on small datasets or with limited parameters. Even though these results are preliminary and obtained only on two datasets one of viral classes and another of human genes subject to polynucleotide expansion (which we believe at this moment is the actual limitation of this novel method), we expect entropy to play an increasingly crucial role in the interpretation of genetic information, with all the resulting benefits in the medical and biological fields. Future work could include implementing these techniques in the field of pathological polynucleotide repetitions or creating compact devices that run very lightweight neural networks for classification. We look forward to applying these methods in the study of rare genetic diseases, which we believe could be the major impact of our work.

6 Data availability

The code used for the benchmarks is available in the GitHub repository [15] along with a dataset of 24 human DNA sequences subject to polynucleotide expansion (pathological and not). The reference dataset for viral DNA sequences is taken from [14], consisting of 6 viral classes, with a training CSV file containing 1320 sequences, a development set with 180 sequences, and a test dataset with 400 sequences. The codes were written in Python using the PyTorch library[34] that allows GPU hardware acceleration with CUDA[35]. Values are macro-averages over 40 random restarts.
The machine used for running the code has the following components:
2 Intel Xeon Gold 6338
1024 GB DDR4 3200 ECC Registered RAM
Intel C621A Chipset
16 TB SATA/600 HDD (7200 rpm)
2.5" 7.68 TB NVMe PCIe 4.0 SSD
4 GPU RTX A5000 with 24 GB VRAM (230 W each)
OS: Ubuntu Server 24.04

7 Competing interests

No competing interest is declared.

8 Acknowledgments

The authors gratefully acknowledge the Department of Biology and Earth Science at the University of Calabria for providing access to their high-performance workstation, which was instrumental in conducting the computational experiments. The authors thank Dr. Gabriel Antonio Videtta for his valuable suggestions in proving Theorem 1.

References

[1] Watson, J. D.; Crick, F. H. C.
Molecular Structure of Nucleic Acids: A Structure for Deoxyribose Nucleic Acid.
Nature **1953,** *171*, 737–738.
DOI: 10.1038/171737a0.

[2] Nirenberg, M. W.; Matthaei, J. H.
The Dependence of Cell-Free Protein Synthesis on Naturally Occurring or Synthetic

Polyribonucleotides.
Proc. Natl. Acad. Sci. USA **1961,** *47*(10), 1588–1602.
DOI: 10.1073/pnas.47.10.1588.

[3] Gatlin, L. L.
The Information Content of DNA.
Journal of Theoretical Biology **1966,** *10*(2), 281–300.
DOI: 10.1016/0022-5193(66)90127-5.

[4] Schneider, T. D.; Stormo, G. D.; Gold, L.; Ehrenfeucht, A.
Information Content of Binding Sites on Nucleotide Sequences.
Journal of Molecular Biology **1986,** *188*(3), 415–431.
DOI: 10.1016/0022-2836(86)90165-8.

[5] Shannon, C. E.
A Mathematical Theory of Communication.
Bell Syst. Tech. J. **1948,** *27*, 379–423 (and 623–656).
DOI: 10.1002/j.1538-7305.1948.tb01338.x.

[6] Koslicki, D.
Topological Entropy of DNA Sequences.
Bioinformatics **2011,** *27*(8), 1061–1067.
DOI: 10.1093/bioinformatics/btr077.

[7] Schmitt, A. O.; Herzel, H.
Estimating the Entropy of DNA Sequences.
Journal of Theoretical Biology **1997,** *188*(3), 369–377.
DOI: 10.1006/jtbi.1997.0493.

[8] Angermueller, C.; Pärnamaa, T.; Parts, L.; Stegle, O.
Deep Learning for Computational Biology.
Mol. Syst. Biol. **2016,** *12*, 878.
DOI: 10.15252/msb.20156651.

[9] Eraslan, G.; Avsec, Ž.; Gagneur, J.; et al.
Deep Learning: New Computational Modelling Techniques for Genomics.
Nat. Rev. Genet. **2019,** *20*, 389–403.
DOI: 10.1038/s41576-019-0122-6.

[10] Ching, T.; Himmelstein, D. S.; Beaulieu-Jones, B. K.; Kalinin, A. A.; Do, B. T.; Way, G. P.; et al.
Opportunities and Obstacles for Deep Learning in Biology and Medicine.
J. R. Soc. Interface **2018,** *15*(141), 20170387.
DOI: 10.1098/rsif.2017.0387.

[11] Yao, M.
Fuzzy Entropy Based Combined Learning Algorithm for Neural Networks.
J. Syst. Eng. Electron. **1996,** *7*(1), 15–22.

[12] Alipanahi, B.; Delong, A.; Weirauch, M.; et al.
Predicting the Sequence Specificities of DNA- and RNA-Binding Proteins by Deep Learning.
Nat. Biotechnol. **2015,** *33*, 831–838.
DOI: 10.1038/nbt.3300.

[13] Ji, Y.; Zhou, Z.; Liu, H.; Davuluri, R. V.
DNABERT: Pre-trained Bidirectional Encoder Representations from Transformers Model for DNA-Language in Genome.
Bioinformatics **2021,** *37*(15), 2112–2120.
DOI: 10.1093/bioinformatics/btab083.

[14] Zolfaghari, A.
DNA Sequence Classification Dataset.
GitHub Repository, 2025.
Available at: `https://github.com/arminZolfaghari/DNA-Sequence-Classification/tree/main/Dataset` (Accessed: 1 March 2025).

[15] Pastore, E. P.
Benchmark codes for CNN DNA-classifiers that use different complexity-based cropping techniques.
Zenodo Repository, 2025.
Available at: `https://doi.org/10.5281/zenodo.15399359` (Accessed: 13 May 2025).

[16] Tenreiro Machado, J. A.; Costa, A. C.; Quelhas, M. D.
Entropy Analysis of the DNA Code Dynamics in Human Chromosomes.
Computers & Mathematics with Applications **2011,** *62*(3), 1612–1617.
DOI: 10.1016/j.camwa.2011.03.005.

[17] Jin, S.; Tan, R.; Jiang, Q.; Xu, L.; Peng, J.; Wang, Y.; Wang, Y.
A Generalized Topological Entropy for Analyzing the Complexity of DNA Sequences.
PLoS ONE **2014,** 9, e88519.
DOI: 10.1371/journal.pone.0088519.

[18] Loewenstern, D.; Yianilos, P. N.
Significantly Lower Entropy Estimates for Natural DNA Sequences.
J. Comput. Biol. **1999,** *6*(1), 125–142.
DOI: 10.1089/cmb.1999.6.125.

[19] Dorier.
A General Outline of the Genesis of Vector Space Theory.
Historia Mathematica **1995,** *22*(3), 227–261.
DOI: 10.1006/hmat.1995.1024.

[20] Marcais, G.; Kingsford, C.
A Fast, Lock-Free Approach for Efficient Parallel Counting of Occurrences of k-mers.
Bioinformatics **2011,** *27*(6), 764–770.
DOI: 10.1093/bioinformatics/btr011.

[21] Berkes, I.; Philipp, W.; Tichy, R.
Entropy Conditions for Subsequences of Random Variables with Applications to Empirical Processes.
Monatshefte für Mathematik **2008,** *153*, 183–204.
DOI: 10.1007/s00605-007-0519-8.

[22] Farach-Colton, M.; Noordewier, M.; Savari, S.; Shepp, L.; Wyner, A.; Ziv, J.
On the Entropy of DNA: Algorithms and Measurements Based on Memory and Rapid Convergence.
In: *Proc. Sixth Annu. ACM-SIAM Symp. Discrete Algorithms* **1995,** 48–57.
DOI: 10.1145/313651.313662.

[23] Biswas, S.; Sarkar, B. K.
Entropy of DNA Sequences as Similarity Index for Various SARS-CoV-2 Virus Strains.
In: *Advances in Medical Physics and Healthcare Engineering*, Lecture Notes in Bioengineering, edited by Mukherjee, M.; Mandal, J.; Bhattacharyya, S.; Huck, C.; Biswas, S., Springer, Singapore, 2021.
DOI: 10.1007/978-981-33-6915-3_51.

[24] Janson, S.; Lonardi, S.; Szpankowski, W.
On the Average Sequence Complexity.
In: *Combinatorial Pattern Matching (CPM 2004), Lecture Notes in Computer Science*, edited by Sahinalp, S. C.; Muthukrishnan, S.; Dogrusoz, U., vol. 3109, pp. 63–76, Springer, Berlin, Heidelberg, 2004.
DOI: 10.1007/978-3-540-27801-6_6.

[25] Ebeling, W.; Jimenez-Montano, M.; Pohl, T.
Entropy and Complexity of Sequences.
In: Karmeshu (Ed.), *Entropy Measures, Maximum Entropy Principle and Emerging Applications*, Studies in Fuzziness and Soft Computing, vol. 119, pp. 233–262, Springer, Berlin, Heidelberg, 2003.
DOI: 10.1007/978-3-540-36212-8_11.

[26] Tsallis, C.
Possible Generalization of Boltzmann–Gibbs Statistics.
J. Stat. Phys. **1988,** *52*(1–2), 479–487.
DOI: 10.1007/BF01016429.

[27] Huang, W.; Maass, A.; Ye, X.
Sequence Entropy Pairs and Complexity Pairs for a Measure.
Ann. Inst. Fourier **2004,** *54*(4), 1005–1028.
DOI: 10.5802/aif.2041.

[28] Bakouch, H. S.; Aghababaei Jazi, M.; Nadarajah, S.
A New Discrete Distribution.
Statistics: A Journal of Theoretical and Applied Statistics **2012,** *48*(1), 1–41.
DOI: 10.1080/02331888.2012.716677.

[29] Stanley, R. P.
Enumerative Combinatorics: Volume 1.
Cambridge University Press, 2011.

[30] Debnath, L.
Srinivasa Ramanujan (1887–1920) and the Theory of Partitions of Numbers and Statistical Mechanics: A Centennial Tribute.
Int. J. Math. Math. Sci. **1987.**
DOI: 10.1155/S0161171287000772.

[31] Ho, S.-W.; Verdú, S.
Convexity/Concavity of Rényi Entropy and α-Mutual Information.
In: *2015 IEEE International Symposium on Information Theory (ISIT)*, Hong Kong, China, 2015, pp. 745–749.
DOI: 10.1109/ISIT.2015.7282554.

[32] Lockwood, E.
The Strategy of Solving Smaller, Similar Problems in the Context of Combinatorial Enumeration.
Int. J. Res. Undergrad. Math. Ed. **2015,** 1, 339–362.
DOI: 10.1007/s40753-015-0016-8.

[33] Li, W.; Freudenberg, J.; Miramontes, P.
Diminishing Return for Increased Mappability with Longer Sequencing Reads: Implications of the k-mer Distributions in the Human Genome.
BMC Bioinformatics **2014,** 15, 2.
DOI: 10.1186/1471-2105-15-2.

[34] Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al.
PyTorch: An Imperative Style, High-Performance Deep Learning Library.
In: *Advances in Neural Information Processing Systems 32*, 2019, pp. 8024–8035.
DOI: 10.48550/arXiv.1912.01703.

[35] *CUDA Compatible GPU Cards as Efficient Hardware Accelerators for Smith–Waterman Sequence Alignment.*
BMC Bioinformatics **2008,** 9(S2), S10.
DOI: 10.1186/1471-2105-9-S2-S10.