Spring 5-6-2024

Designing Methods for Representation Learning of Molecular Sequences and its Application in Analysis Tasks

Taslim Murad

Designing Methods for Representation Learning of Molecular Sequences and its Application in Analysis Tasks

by

Taslim Murad

Under the Direction of Murray Patterson, Ph.D.

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

in the College of Arts and Sciences

Georgia State University

2024

ABSTRACT

Molecular sequence analysis serves as a fundamental process for elucidating the intricate functions, structures, and behaviors inherent in sequences. Its application extends to characterizing associated organisms, such as viruses, facilitating the development of preventive measures to mitigate their dissemination and influence. Given the potential of viruses to trigger epidemics with global ramifications, comprehensive sequence analysis is pivotal in understanding and managing their impact effectively. The rapid expansion of bio-sequence data has surpassed the computational capabilities of traditional analytical techniques, such as the phylogenetic approach, due to their high computational costs. Consequently, clustering and classification have emerged as compelling alternatives, with machine learning (ML) and deep learning (DL) algorithms capable of effectively implementing these methods. Although ML/DL models are known for their high analytical capabilities, however, they typically require the inputs to be either in numerical or image form. Therefore, efficient and effective mechanisms are needed to transform bio-sequences into ML/DL-compatible inputs, and this research intends to devise such techniques. In this regard, alignment-free and fast feature-engineering-based approaches and image-based approaches are put forward in this work to convert the bio-sequences into numerical and image form respectively. The feature-engineering-based methods, PSSMFreq2Vec and PSSM2Vec combine the power of k-mers and position weight matrix (PWM) to be scalable, alignment-free, and compact, while Hashing2Vec utilizes the combination of hashing and k-mers to achieve high embedding generation speed and to be alignment-free respectively. Furthermore, two of the image-based approaches follow the underlying concept of Chaos Game Representation (CGR) to map sequences to images while one uses Bezier function-based mapping of sequences into images, and they aim to enable the application of sophisticated vision DL analytical models on bio-sequences. The representations gained from both feature-engineering-based and image-based

methods are passed on to ML/DL models to perform classification tasks and their results illustrate high predictive performance as compared to the respective baseline models.

INDEX WORDS: Biological Sequence Analysis, Feature-Engineering-based Bio-Sequence Representation, Image Classification, Machine Learning, Deep Learning, Chaos Game Representation, Bezier Function

Designing Methods for Representation Learning of Molecular Sequences and its Application in Analysis Tasks

by

Taslim Murad

Committee Chair: Murray Patterson

Committee: Alex Zelikovsky

Esra Akbas

Daniel Takabi

Zhisheng Yan

Electronic Version Approved:

Office of Graduate Services

College of Arts and Sciences

Georgia State University

May 2024

DEDICATION

I dedicate this work to everyone in my life who has been a support so far, especially my parents, my siblings and my husband Karim. Their unconditional support, encouragement and love has made this possible.

ACKNOWLEDGMENTS

I thank my advisor, committee members and lab mates for all the help they have offered during my PhD journey. I also extend my gratitude to my dear friends and roommates (Inara, Rabia & Samana) for always being there in any situation. I also want to thank my friend Mild for being so supportive and patient.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
Introduction

Molecular sequences usually refer to nucleotide or amino acids sequences. Their analysis can provide detailed information about the functional and structural behaviors of the corresponding organisms, like viruses etc. In this study, we focus on the analysis of viruses, which are usually responsible for causing diseases for example Flu Das (2012), Covid-19 Pedersen et al. (2020), etc. The genetic diversity and dynamics (e.g., mutations, variations, hosts) of a virus can also be investigated through analysis. Gaining a deeper understanding of a virus is very helpful in building prevention mechanisms, like drugs Rognan (2007), vaccines Dong & Pei (2007), etc., to control its associated disease spread and eliminate the negative impacts. It can also be useful in virus-spread surveillance.

1.1 Motivation

The Severe Acute Respiratory Syndrome Coronavirus 2 (SARS-CoV-2) virus is such an example that has caused COVID-19 disease. This disease has infected almost 1.36 million people from 219 countries as of April 2021 Uyangodage et al. (2021). According to a recent report (June 2022) by the Centers for Disease Control and Prevention (CDC), a total of $86,379,937$ cases are reported in the United States alone. With the pandemic levels of COVID-19, an unprecedented amount of SARS-CoV-2 genomic sequencing data has been collected and is still ongoing. Such data is essential for comprehending the disease, which will help researchers to advise preventive measures and minimize its effects. Moreover, it is well-known that many major mutations happen in the spike region of the SARS-CoV-2

genome Kuzmin et al. (2020); Ali et al. (2021). It can be due to the role of this region to attach the virus to the host cell membrane Kuzmin et al. (2020). Thus the spike protein region provides sufficient information about the virus's genome that can be used to perform further analysis of the virus. The structure of the SARS-CoV-2 genome is shown in Figure 1.1. It is approximately 30kb in length, the spike region lying in the 21–25kb range, coding for a "spike" protein of 1273 amino acids, which can be divided into two sub-units $S1$ and $S2$. The receptor-binding domain (RBD) is a region in the subunit $S1$ which ranges from amino acids 319 to 541 of the spike protein. To infect the host cell, the spike protein RBD of the SARS-CoV-2 virus engages with the ACE2 cell surface protein. SARS-CoV-2 variants of concern that have appeared independently across the world feature mutations in the RBD Morales et al. (2021). Thus the spike sequence is often sufficient for characterizing a given variant of SARS-CoV-2 Kuzmin et al. (2020). Furthermore, the SARS-CoV-2 virus has been studied recently by analyzing its spike sequences, which includes performing classification tasks using these sequences. These tasks are either host-wise Kuzmin et al. (2020); Ali et al. (2022) or variant-wise Ali et al. (2021); Ali & Patterson (2021); Tayebi et al. (2021). Host-wise intends to classify the sequences based on the corresponding infected hosts, while variant-wise follows the virus's variant/lineage-based classification. The high performance of ML/DL classification models makes them an ideal choice for spike protein sequence classification, however, these sequences need to be transformed into ML/DL compatible inputs first to enable the application of ML/DL models.

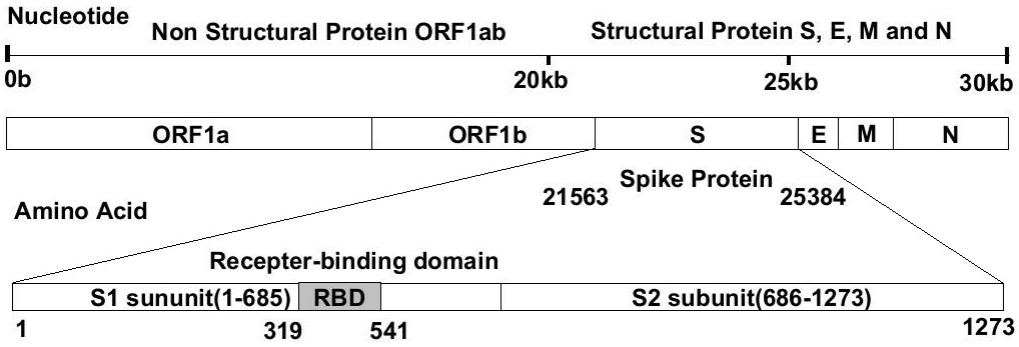


Figure 1.1 The SARS-CoV-2 genome, roughly 30kb in length, codes for both structural and non-structural proteins. Among the structural proteins, S, E, M, and N, the S (spike) protein is responsible for the attachment of the virus to the host cell membrane. Advantageous mutations in the spike region are often responsible for increased transmissibility of the virus.

1.2 Bio-Sequence Embedding Generation Techniques

There are many methods put forward in literature to produce ML/DL compatible transformation of spike sequences, like one-hot encoding (OHE) Kuzmin et al. (2020). It yields a binary feature vector of a given spike sequence, however, it faces the curse of dimensionality and sparsity challenges. Similarly, a k-mer (see Definition 1) based alignment-free approach Ali & Patterson (2021) is proposed, but it has very high embedding generation time complexity making it difficult to be scalable and it also undergoes the sparsity issue. Likewise, a position weight matrix (PWM) based encoding method Ali et al. (2022) is given to obtain the numerical encoding of the sequences, however, this method is alignment-based and sequence alignment is a computationally expensive process. Sequence alignment is a way of arranging the sequences of DNA, RNA, or protein to identify regions of similarity that may be a consequence of functional, structural, or evolutionary relationships between the sequences. Futhemore, kernel matrix based method Farhan et al. (2017) is proposed

but it is space inefficient. Moreover, to enable the application of sophisticated DL vision classification models on bio-sequence data, the algorithms to convert sequences into images following the CGR Jeffrey (1990) concept are presented. Like Hoang et al. (2016); Rizzo et al. (2016) proposed CGR-based algorithms to build images of DNA (nucleotide-based) sequences, however, applying them on spike protein sequences causes overlapping in the corresponding generated images. Authors of Löchel et al. (2020) proposed modifications to CGR known as FCGR (Frequency Chaos Game Representation) to handle protein sequences. However, they consider the amino acids of a sequence one by one (rather than considering substrings using n-gram Ali & Patterson (2021); Ali et al. (2021)) and assign equal weight to every amino acid (i.e., every pixel in the image corresponding to an amino acid will have a value 1 rather than weights based on their positions in the sequence) in their respective image representation.

Definition 1 (k-mers)**.** *It is referred to as a set of (consecutive) amino acids of length k for any given sequence (also called nGram in the NLP domain). The total number of k-mers generated for a sequence of length N is $N - k + 1$.*

Therefore, this work deals with proposing alignment-free and time-efficient mechanisms to transform spike protein sequences into ML/DL-compatible inputs. Some of these methods are shown to be scalable, while others enable the application of DL classifiers on spike protein sequences. The proposed work is categorized into two groups and they are as follows,

1.2.1 Feature-Engineering-based Encoding Methods

Three alignment-free numerical embedding generation algorithms for spike protein sequences are proposed in this work. These algorithms are PSSMFreq2Vec, PSSM2vec, and Hashing2Vec. The PSSMFreq2Vec and PSSM2vec techniques combine the power of k-mers and PWM to be scalable and alignment-free. They also demonstrate compactness as compared to their respective baselines. Hashing2Vec merges the underlying concepts of k-mers and hashing to be alignment-free and fast. The numerical features obtained from these encoding methods are further given to ML classifiers to perform classification tasks.

1.2.2 Image-based Encoding Methods

Image classification is a well-studied problem. Many sophisticated DL models have been developed in past years to achieve state-of-the-art performance in terms of image classification using DL architectures such as convolutional neural networks (CNN). To enable the application of high-performing DL vision classifiers on bio-sequences, we proposed three methods to convert sequences into images. Two of them are based on the concept of chaos game representation, while one follows Bezier curve function to generate images. Some advantages of converting sequences to images are that these approaches does not depend on sequence length or alignment (the size of the image remains fixed) and they enable DL

classifiers application on the spike protein classification.

1.3 Contributions

The contributions made by this work are the following,

1. Three alignment-free, fast, and scalable numerical embedding generations algorithms (PSSMFreq2Vec, PSSM2Vec, Hashing2Vec) are proposed to convert the SARS-CoV-2 spike protein sequences into numerical form to enable ML-based classification on them.

2. An image-based method, named Spike2CGR, is put forward to generate the images of spike protein sequences of the SARS-CoV-2 virus based on the concepts of chaos game representation (CGR) and minimizers.

3. A novel approach for converting biological sequences into images utilizing the Bézier function is also presented. By harnessing the capabilities of the Bézier curve in conjunction with deep learning analytical models, we can foster a more profound comprehension of these sequences. This innovative technique holds promise for advancing our understanding of biological data and enabling more robust analysis and insights.

4. A new image-based approach for ACP classification is proposed which explore the use of secant and cosecant functions as an alternative to sine and cosine functions in the CGR technique to generate a more rectangular mapping. We also combined spaced k-mers concept usage with the CGR method to produce better images in terms of predictive performance.

5. The statistical analysis and visualization of feature-engineering methods are provided.

6. The statistical analysis of image-based encoding methods is discussed.

CHAPTER 2

Background

Molecular sequence study is a popular topic in research, like protein analysis Rao et al. (2014); Buchan & Jones (2019) is essential for inferring its functional and structural properties, which helps in understanding diseases and building prevention mechanisms like drug discovery, etc. In this regard, learning-based methods play a vital role and they usually consist of an essential step of representation learning. Various representation learning-based techniques are presented to perform bio-sequence analysis using learning methods and they are categorized as follows,

2.1 Feature-Engineering-based Analysis

Identifying sequence homology (common ancestry) between proteins and predicting disease transmission using phylogeny-based techniques Dhar et al. (2020) are crucial problems in bioinformatics. However, the millions of sequences now accessible for viruses, like SARS-CoV-2, greatly exceed the capacity of such phylogeny techniques Hadfield et al. (2018); Minh et al. (2020). Various feature embedding-based methods are put forward to gain a deeper understanding of the biological sequences, like Kuzmin et al. (2020) proposed a one-hot encoding (OHE) technique to classify spike protein sequences of SARS-CoV-2 virus by transforming the sequences into binary vectors. However, this technique undergoes the curse of dimensionality and sparsity challenges, and it doesn't retain any ordering information of the sequence. Similarly, a k-mer-based alignment-free approach, known as Spike2Vec Ali & Patterson (2021), is proposed to perform spike protein analysis, but it has very high

embedding generation time complexity making it difficult to be scalable and it also undergoes the sparsity issue. Likewise, a position weight matrix (PWM) based encoding method PWM2Vec Ali et al. (2022) is given to obtain the numerical encoding of the spike sequences, however, this method is alignment-based and sequence alignment is a computationally expensive process. PWKmer Ma et al. (2020) method uses position distribution information and k-mers frequencies to do a phylogenetic analysis of HIV-1 viruses. However, these methods are computationally expensive and can face the curse of dimensionality challenge.

2.2 Neural Network-based Analysis

Moreover, another set of bio-sequence analysis mechanisms employs neural networks to generate numerical representations like WDGRL Shen et al. (2018), AutoEncoder Xie et al. (2016) etc. WDGRL is an unsupervised technique that uses a neural network to extract numerical embeddings from the sequences. AutoEncoder follows the encoder-decoder architecture and the encoder network yields the feature embeddings for any given sequence. These embeddings are used to do ML-based classification. However, these methods are computationally expensive and require large training data to achieve good performance.

Furthermore, a set of pre-trained models to deal with protein classification are also introduced like Protein Bert Brandes et al. (2022), Seqvec Heinzinger et al. (2019), UDSM-Prot Strodthoff et al. (2020) etc. In Protein Bert an NN model is trained using protein sequences and this pre-trained model can be employed to get embeddings for new sequences. Likewise, SeqVec provides a pre-trained deep language model for generating protein se-

quence embeddings. In UDSMProt a universal deep sequence model is put forward which is pre-trained on unlabeled protein sequences from Swiss-Prot and further fine-tuned for the classification of proteins. However, all these methods have heavy computational costs.

2.3 Kernel-based Analysis

Several kernel-based analysis techniques are proposed, like Farhan et al. (2017) which computes the distance between sequences using the number of matches and mismatches between characters (amino acids) from k-mers and designs the kernel (or gram) matrix. But its memory inefficient because it has to load the entire kernel matrix (which is very high dimensional) in the memory. Another method is gapped k-mer (Gkm) string kernel Ghandi et al. (2014) which enables the usage of string inputs (biological sequences) for training SVMs. It determines the similarity between pairs of sequences using gapped k-mers, which eradicates the sparsity challenge associated with k-mers. However, the interpretation of gkmSVMs can be challenging. GkmExplain Shrikumar et al. (2019) is an extension of Gkm which claims to be more efficient in performance. The string kernel Lodhi et al. (2002) is a kernel function that is based on the alignment of substrings in sequences but it's space inefficient.

2.4 Image-based Analysis

An alternative molecular sequence analysis category contains the methods which transform the sequences into images to perform further analysis. These methods follow CGR which focuses on the visual encoding of sequences based on generating fractals. Although CGR builds a visual representation of a given sequence, it was originally designed for DNA (nucleotide)

sequences Jeffrey (1990); Hoang et al. (2016); Rizzo et al. (2016). Authors of Löchel et al. (2020) proposed modifications to CGR known as FCGR (Frequency Chaos Game Representation) to handle protein sequences. However, they consider the amino acids of a sequence one by one (rather than considering substrings using n-gram Ali & Patterson (2021); Ali et al. (2021)) and assign equal weight to every amino acid (i.e., every pixel in the image corresponding to an amino acid will have a value 1 rather than weights based on their positions in the sequence) in their respective image representation. Moreover, the existing CGR-based approaches operate on k-mer (for $k > 1$) for nucleotide only and only 1-mers in the case of protein sequences (also known as FCGR). However, better underlying sequence representations, such as Minimizers, can be used rather than k-mers. Similarly, using $k > 1$ for FCGR could also produce better results.

CHAPTER 3

Proposed Approaches

This work consists of two categories of proposed methods to convert spike protein sequences of the SARS-CoV-2 virus into ML/DL-compatible forms. A detailed discussion of each method is mentioned below,

3.1 Feature Engineering Methods

Three different methods are designed to generate numerical feature embeddings from the spike sequences. These embeddings are further utilized by the ML models to perform the classification of the sequences. The methods are as follows,

3.1.1 PSSMFreq2Vec

For a given spike sequence, PSSMFreq2Vec works by using the idea of k-mers (Defination 1) to design a PWM and then assign a weighted value to each k-mer based on the values for different amino acids at different positions in the PWM. The position weight matrix (PWM) Stormo et al. (1982), also known as the position-specific scoring matrix (PSSM), is utilized historically for the portrayal of biological sequences-based motifs (patterns). It comprises information about the count of amino acids for each position in the form of weights (log-likelihood). Then a fixed-length feature vector for all possible combinations of amino acids of length k is generated and the score from the PWM to the respective k-mer bin is added. All remaining entries of the vector will have a value of zero. In this way, the locality

information is captured by using the k-mers, and the importance of different characters' positions in the sequence is also computed. Such weighted information cannot be computed by using k-mers only. Moreover, as PSSMFreq2Vec is a (weighted) frequency vector, it does not rely on a global alignment so it is alignment-free. Combining these pieces of information in this way allows us to devise a compact, general, and alignment-free feature embedding technique, which can convert many types of bio-sequence data to numerical form so that the data is compatible with many different downstream ML tasks.

A summary of the PSSMFreq2Vec algorithm is shown in Figure 3.1. It follows the given steps (from a to h) for generating a feature vector against a sequence. In step (a), a spike protein sequence is provided, which is used to extract k-mers in (b) ($k = 3$ is used in experiments). Further, in step (c), a position frequency matrix (PFM) is generated from these k-mers, which stores the count of each character of their position in the k-mers. The dataset used in the experiments contains 20 unique characters (represented by Σ, the set of amino acids) with $k = 3$, so the dimensions of PFM, in this case, is 20×3. In step (d), a position probability matrix (PPM) is generated by converting PFM values into column-wise probabilities. To obtain the PPM, we divide the count for a given character by the total count of characters in the column. To avoid zero values in the PPM, a Laplace value (pseudo count) is added to every element, resulting in the matrix depicted in (e). A Laplace value of 0.1 is used in the experiments Nishida et al. (2009). In (f), a position weight matrix (PWM) is produced from the Laplace-adjusted PPM by computing the log-likelihood of each character $c \in \Sigma$ at a position i with the formula $W_{c,i} = \log_2(p(c,i)/p(c))$, where $p(c)$ is

defined as $n(c)/61$ and $n(c)$ is the number of codons for each amino acid $c \in \Sigma$ and 61 is the number of sense codons. Then the absolute scores of each k-mer is calculated by summing up the individual scores from the PWM of every k-mer character with respect to their position in the PWM, resulting in the vector (g). In the last step (h), a feature vector of length $|\Sigma|^k$ is created, representing all the possible k-combinations of Σ. The feature vector consists of all zero values except for the positions representing the k-mers, which hold the absolute scores (computed in (g)) of the respective k-mers. The entire process is repeated for each spike sequence.

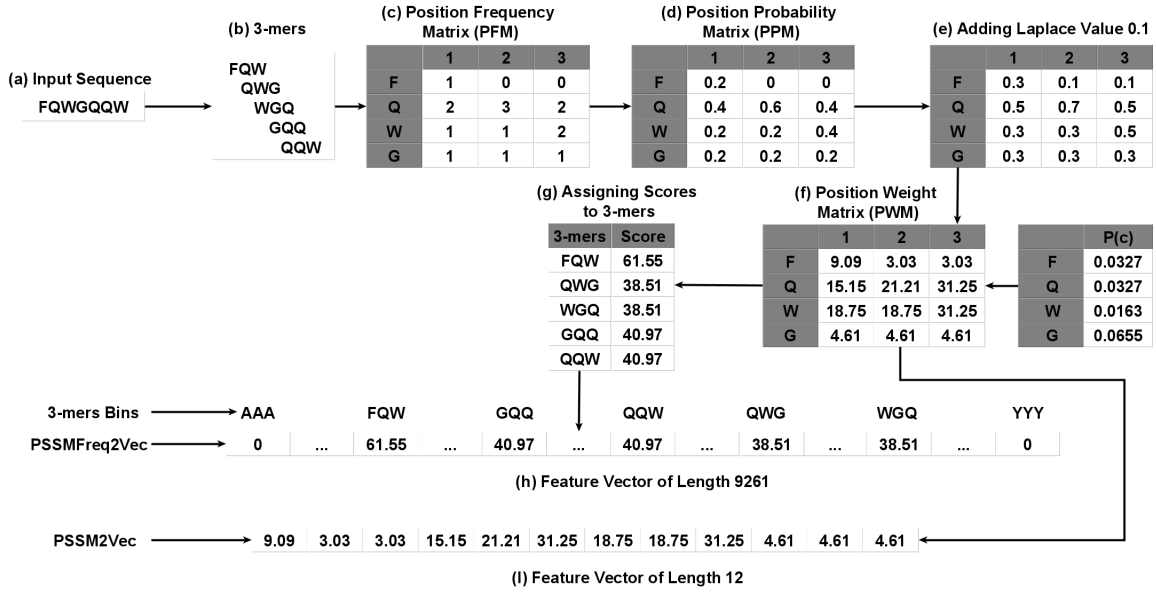


Figure 3.1 PSSMFreq2Vec and PSSM2Vec flow chart. For PSSMFreq2Vec, a feature vector is built from a sequence by computing PWM from k-mers, creating a zero feature vector of length $|\Sigma|^k$, and updating its values accordingly. For PSSM2Vec, the vector is built by flattening the PWM matrix in step (f).

3.1.2 PSSM2Vec

This embedding approach is a more compact and generalized version of PWM2Vec Ali et al. (2022). Given a sequence, PSSM2Vec first designs the position weight matrix (PWM) and then "flattens" (concatenates the rows) to obtain the resulting feature vector. The flowchart for PSSM2Vec is given in Figure 3.1. Note that the steps from (a) to (f) are the same as in PSSMFreq2Vec, but in the last step (I), the PWM is simply flattened to obtain a feature vector corresponding to the input sequence. Again, this process is repeated for each protein sequence.

3.1.3 Hashing2Vec

This is another numerical embedding (ϕ) generation strategy for spike sequences. Let Σ represent the alphabet (the set of all unique amino acids comprised of $ACDEFGHIKLMNPQRSTVWXY$) and k is the length of a k-mer. The total number of k-mers of length k created from the given spike sequence will be $|\Sigma|^k$. The Hashing2Vec model uses a hashing technique with a hash table of size m to obtain embeddings and it reduces the bin searching overhead (see Definition 2).

Moreover, the Algorithm 1 demonstrates the pseudocode for the overall pipeline of the Hashing2Vec method. We can observe that, for a given sequence s, k-mer length k, and hash table size m, it returns the *exact* feature embedding of the sequence. The first step involves computing unique k-mers of size k in a spike sequence of length n. Then a dictionary (local hash value to k-mers within a spike sequence) of size d is created, where $d \leq n-k+1$ (since

there will be repetitive k-mer in a sequence) and store the counts of each unique k-mer in the dictionary. After that, the (global) hash value for all possible k-mers in the data is computed, and a hash table position is assigned to them. For each (local) k-mer within a spike sequence, we use its global hash value and place its count in the hash table (we use this hash table as a feature embedding ϕ). The dimension of ϕ is the size m of the hash table. The final step is to use Principal Component Analyses (PCA) to get a low dimensional representation of ϕ. Each step is further explained in more detail below:

Definition 2 (Bin searching)**.** *Given a vector for all possible k-mers ($|\Sigma^k|$), each k-mer in the spike sequence is assigned to a bin in the feature vector. Since we do not (initially) know the position of the bin for a specific k-mer, we need to perform searching. This problem is called bin searching.*

3.1.3.1 Step 1: Generating k-mers:

To generate the fixed-length embedding for a given spike sequence, generate its all possible k-mers. An example of generating k-mers for a given spike sequence is shown in Figure 3.1(c). For Hashing2Vec embedding generation, we took $k = 3$, which is decided using standard validation set approach Devijver & Kittler (1982).

3.1.3.2 Step 2: Counting the k-mers:

After creating the k-mers, count the number of each k-mer by storing the unique k-mers

in a dictionary (this can be thought of as "local" hashing of the k-mers within a specific spike sequence). After getting the k-mers count, the next step is to design an embedding ϕ, where each unique k-mer is assigned a bin that will contain its count as the numerical value. The k-mers that are not present in a given spike sequence will have zero value in ϕ. To find the optimal bin for the k-mers in ϕ, a brute-force method is used to search the embedding to see which bin a specific k-mer belongs to. For each k-mer in a given spike sequence, this step must be repeated. In the worst case, this *bin search* for the relevant k-mers position can end up being an expensive process. Therefore, we intend to solve the problem of bin searching (see Definition 2) of ϕ in this work, after the k-mers frequencies computation is done. Note that bin searching is not an optimization problem. Note that we are not concerned with the k-mers counting algorithm, rather our focus is to improve the ϕ generation as a result of k-mers counting (i.e., improving the traditional bin searching mechanism). There are many efficient and fast methods for k-mers counting, however, discussing (and using) those methods are out of the scope of this research work.

3.1.3.3 Step 3: Assign Unique ID to k-mers Using Hashing:

The *bin searching* problem is solved by utilizing the hashing technique in this work. More specifically, the k-mers are hashed (refer to them as "Global" hash values) using a popular hash function, called Fowler–Noll–Vo Fowler et al. (2011), to assign a unique ID (hash table entry) to each k-mer. FNV (FNV-1a 32bit specifically) works by initializing a hash variable. Then it performs two major operations for each byte of data. First, an XOR of the byte and

the hash is performed, and then the result is multiplied by a particular prime number. After mapping the k-mers to consistent hash values, we designed a frequency-based feature vector. For a given spike sequence, we now have a k-mers count (computed in step 2) and a global hash value (computed using FNV). At the global hash table entry, we place the k-mer count directly for all k-mers in a given spike sequence. All the other entries in the hash table will have the value 0. The experiments show that for $k = 3$, the optimal hash table size (m) is 404048. The optimality of m is determined (iteratively) by eliminating the collisions of hash values. Since the parameter m (hash table size) is a learned parameter, hence it guarantees zero collisions in the hash table. The value of m is learned by iteratively increasing the hash table size until each unique k-mer gets a unique hash table id. Therefore our created feature embeddings are exact (not approximate embeddings). Moreover, the learning of m needed to be done only once in the start, and we can then have $O(1)$ time complexity for placing each k-mers in relevant bins of feature vector (the hash table entry) afterward for any number of sequences (hence it could be scaled to any size of data as no bin searching overhead is required). For Hashing2Vec, we only require one hash function. The resultant hash table (containing k-mers count) is considered as the final feature vector ϕ. At this point, for q spike sequences (total number of spike sequences in the data), we get a $q \times m$ dimensional matrix, where each row corresponds to a numerical representation of a spike sequence and each column corresponds to a specific k-mer count. Note that the hash values assigned in Step 3 are different from the dictionary of k-mers discussed in Step 2 (k-mers counting). In step 2, we are interested in designing a dictionary to locally count the unique k-mers within

a specific spike sequence (which will be different for different spike sequences). Now, in order to design a general-purpose feature embedding for a spike sequence, we need a "global" hash value for each possible k-mer in all spike sequences, so that any k-mer in any given spike sequence is mapped to the same hash table entry (the resultant feature vector).

Definition 3 (Fowler–Noll–Vo (FNV))**.** *FNV is a non-cryptographic commonly utilized deterministic hash function.*

3.1.3.4 Step 4: Low Dimensional Representation Using PCA:

Since the dimensionality (size) of the hash table (feature embedding ϕ) is very high, we applied Principle Component Analysis (PCA) Wold et al. (1987) to reduce it. PCA is a popular and widely followed technique for the reduction of data dimensionality. For our experiments, we have chosen the first 500 PCA components.

Furthermore, the workflow of Hashing2Vec is given in Figure 3.2. In the first step, we compute the unique k-mers frequency count for the spike sequence of length n as shown in the left box of Figure 3.2 (a), (b) and (c)). The k-mers are generated using a sliding window of size k. Along with it, a dictionary d is maintained to keep the counts of unique k-mers in a spike sequence, where the size $|d|$ of the dictionary is $|d| \leq n - k + 1$ (since we will have repeated k-mers in the sequence). Afterward, each unique k-mer in the dictionary d is passed through the FNV hash function to get a (global) corresponding hash value. The k-mer frequency count from the dictionary is mapped to the hash table using the respective

computed (global) hash value for each unique k-mer in the dictionary as shown in Figure 3.2 (d)). Finally, the feature embedding (ϕ) of the sequence is generated using the hash table, and the length of the feature embedding (ϕ) is m since the size of the hash table is m.

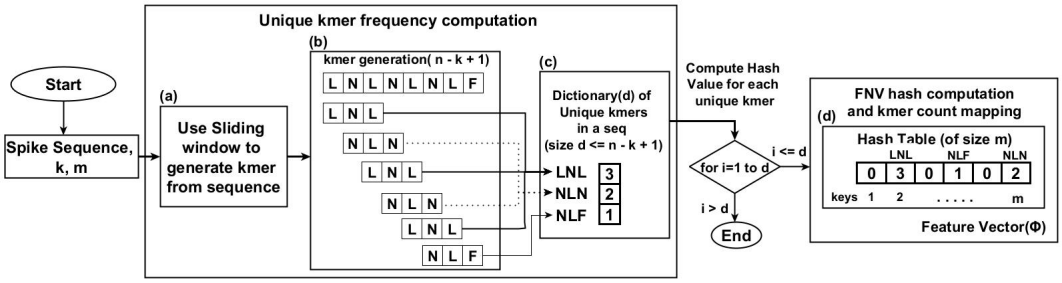


Figure 3.2 Flow chart of Hashing2Vec based embedding.

Algorithm 1 Hashing2Vec

Input: Spike Sequence seq, and integer k and m
Output: Feature Vector ϕ based on Hash Values
$kmers = \emptyset$
ϕ = List(0) × m ▷ feature embedding vector of length m
for $i \leftarrow 1$ to $|seq|$ **do**
    $kmer$.append($seq[i : i + k]$) ▷ create k-mers using sliding window
end for
$unique\_kmer, kmer\_count$ = createDictionary($kmers$); ▷ local hash value
▷ create the FNV hash of size m for each k-mer using its count
for $i \leftarrow 1$ to $|unique\_kmer|$ **do**
    $global\_hash\_value = FNV(unique\_kmer[i], m)$ ▷ map k-mer count to hash table for each unique kmer in dictionary
    $\phi[global\_hash\_value] = kmer\_count[i]$
end for
ϕ = PCA(ϕ, 500); ▷ get the first 500 PCA components
return(ϕ)

3.2 Image-based Methods

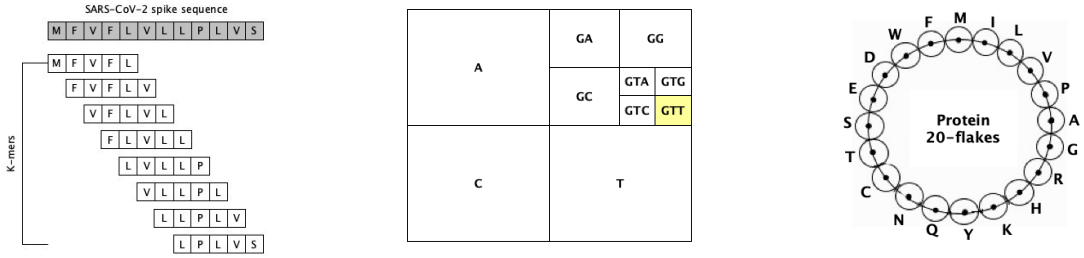
These methods convert the bio-sequences into a image form so that sophisticated vision DL analytical models can be applied to the sequences. The methods are as follow,

3.2.1 Spike2CGR

A popular approach used for encoding biological sequences into images is Chaos Game Representation (CGR) Barnsley (2012); Jeffrey (1990). It was originally designed for nucleotide-based sequences (DNA), and it starts by computing k-mers of a given sequence. After that for each k-mer, its respective nucleotides are utilized to allocate a position to the k-mer in the image. For example, in the case of a DNA genome sequence, an empty image is divided into 4 quadrants, each one representing a unique nucleotide, i.e., A "upper left", C "lower left", G "upper right", and T "lower right". Each of these 4 quadrants is further subdivided, recursively, up until the length k of the k-mer. Based on the nucleotides of a given k-mer, the appropriate position of the k-mer in the image is then detected in this recursive manner, the respective pixel value incremented by 1. This process is visually shown in Figure 3.3b. Although the above-mentioned method works perfectly fine for DNA (genome) sequences (with the number of unique nucleotides $n \leq 4$), it poses the challenge of overlapping in an image for $n > 4$, e.g., protein sequences with twenty proteinogenic amino acids.

Therefore to eliminate the overlapping, authors in Löchel et al. (2020) proposed a frequency matrix-based CGR, known as FCGR (for reference, we call this method "Chaos" in the rest of the report). The Chaos produces an n-flakes (also referred to as poly-flake) Tzanov (2015) based image representation (where n is the number of amino acids), which consists of an image with multiple icosagons (see Figure 3.3c for an example). An n-flake or poly-flake follows an iterative mechanism to construct a fractal starting from an n-gon. For example,

given a spike sequence with 20 amino acids, the Chaos image representation generates twenty edges (one for each amino acid) and then generates twenty inner icosagons within the larger icosagon (see Figure 3.3c).



(a) 5-mers. (b) CGR-based allocation. (c) protein 20-flakes.

Figure 3.3 Example of (a) k-mers, (b) determination of the location (in yellow) of the 3-mer "GTT" in the image using CGR method, and (c) 20-flakes image based on Chaos/FCGR method.

Definition 4 (Icosagons)**.** *In geometry, an icosagon or 20-gon is a twenty-sided polygon (see Figure 3.3c).*

Given a sequence, the fractal is then generated via an iterative process, starting at the center of the image. Each subsequent step is governed by the following process for determining the location of the next pixel in the image based on the previous: First, we calculate the contraction ratio r between the outer and inner polygon. For this purpose, we use the following expression (as proposed in Löchel et al. (2020); Strichartz (2000)):

$$r = \frac{sin(\frac{\pi}{n})}{sin(\frac{\pi}{n}) + sin(\frac{\pi}{n} + \frac{2\pi m}{n})}, \text{for m} = \lfloor \frac{n}{4} \rfloor \tag{3.1}$$

where $n = 20$ for twenty amino acids in the protein/spike sequence. Then we define a scaling factor (SF), which is the ratio of the distance between the current location and the target

edge (amino acid). The SF is computed using the following expression (as proposed in Löchel et al. (2020)):

$$\text{sf} = 1 - r \tag{3.2}$$

Finally, we calculate the coordinates x and y of this next pixel using the following expression (as proposed in Löchel et al. (2020)).

$$x[i] = r \cdot sin(\frac{2\pi i}{n} + \theta) \tag{3.3}$$

$$y[i] = r \cdot cos(\frac{2\pi i}{n} + \theta) \tag{3.4}$$

where $i \in \{1, 2, \ldots, |S|\}$ (where S is a spike sequence and $s[i] \in S$ corresponds to a single amino acid), n is the total number of amino acids, and θ is the angle of orientation. The Chaos method takes a protein sequence as input and yields an image as output by considering the amino acids of the sequence one by one, following Equation 3.3 and Equation 3.4 to get the coordinates of every amino acid in the image. The FCGR generates a greyscale image of a protein sequence based on the coordinates of each amino acid in the spike sequence computed using Equation 3.3 and Equation 3.4. Note that the FCGR allocates pixel value 1 for each amino acid in the spike sequence for which x-y coordinates within the image are computed using Equation 3.3 and Equation 3.4.

In summary, the Chaos method iteration is as follows, starting from point (0,0):

1. Check the next character (amino acid) of the sequence.

2. Go a fraction of the way to the corresponding amino acid (base), according to the

scaling factor.

3. Save coordinates x, y of this next point (and draw a pixel at those coordinates) and repeat (from bullet point 1.).

In this work, four methods are proposed to build images from SARS-CoV-2 spike protein sequences by making different modifications to the Chaos method respectively in order to improve the final predictive performance. Those methods are as follows,

3.2.1.1 Spike2Vec-based Image Encoding

Since the original FCGR (Chaos) includes a single pixel value for each amino acid, it may not represent the spike sequence very effectively. For this purpose, we use a recently proposed method called Spike2Vec Ali & Patterson (2021). The Spike2Vec uses the idea of k-mers to generate the feature embeddings. In this work, we use the same k-mers idea to generate the images from spike sequences by considering a set of amino acids (rather than single amino acid at a time as done by the Chaos approach) with a sliding window (of increment 1) to draw pixels within images. After generating all the k-mers, we concatenate them to make a single (new) sequence, which is used as input to the Chaos method for image generation (using Equation 3.3 and Equation 3.4). In our experiments, the images generated for Spike2Vec use 9-mers (selected using standard validation set approach Devijver & Kittler (1982)).

3.2.1.2 PWM2Vec-based Image Encoding

Both Chaos and Spike2Vec assign an equal weight of 1 to each amino acid. However, this uniform value may not be the most effective way to come up with a image representation of a given sequence. To assign weight to each amino acid within k-mers, we use a recently proposed method, called PWM2Vec Ali et al. (2022). The PWM2Vec technique is also driven by the notion of k-mers, however, instead of using a constant frequency value, it uses weighted values computed from the Position Weight Matrix (PWM) corresponding to alphabets in a sequence. Like Spike2Vec, PWM2Vec enables capturing of locality information due to k-mers usage, but it also considers the importance of relative positions of amino acids in the sequence.

As shown in Figure 3.1, the steps from (a) to (f) are followed to get a PWM for any sequence. After assigning weight to each amino acid within the k-mers of a sequence using PWM2Vec, we use those weights as corresponding pixel values (rather than assigning the pixel value 1 to each amino acid in the sequence). Similar to Spike2Vec, the procedure of PWM2Vec image encoding for a given spike sequence computes k-mers and starts off with a blank image but it includes an additional step of calculating the PWM of the sequence and updating the pixel values corresponding to the amino acids of a k-mer based on PWM values rather than a constant value 1. Now for each k-mer, once the respective coordinates of every amino acid are determined in the image using Equation 3.3 and Equation 3.4, the pixel values representing each amino acid are updated by adding the respective PWM value. Yet again, the task of coordinates finding and pixel value update is repeated for all k-mers

of the given sequence and it results in image generation.

3.2.1.3 Minimizer-based Image Encoding

In this approach, rather than utilizing k-mers for image construction, we use another popular approach in bioinformatics, called minimizers Roberts et al. (2004). One of the main problems with k-mers is that they introduce redundancy (repeated k-mers) in long sequences and hence increase the computation and storage cost. This redundancy could be eliminated using minimizers. The pseudocode of calculating m-mers for any given sequence is shown in Algorithm 2. Its workflow is given in Figure 3.4 and it illustrates that for a given sequence the k-mers of the sequence are extracted (9-mers in the figure example). Then for each k-mer, a minimizer (3-mer in figure example) is computed by getting the lexicographically smallest one from both forward and backward m-mers of the k-mer. The minimizer takes two parameters (k, m). k is the length of k-mer (where $k = 9$ in our case) while m indicates the size of m-mer (where $m = 3$ in our case). Given a spike sequence, it extracts k-mers of the sequence. Then for each k-mer, it computes a corresponding m-mer (minimizer). After computing minimizers, we concatenate all m-mers to make a new sequence and use it as input to the Chaos method for image generation (by computing x and y coordinates of each amino acid in this new sequence using Equation 3.3 and Equation 3.4 and increment the corresponding pixel values of the amino acids by 1). Note that methods like Spike2Vec and PWM2Vec are originally designed to generate numerical vectors. We transform those methods to generate the 2-D visual representations so that we can apply image classification

models for supervised analysis.

Definition 5 (Minimizer)**.** *For a given k-mer, a minimizer (also called m-mer) is a substring of consecutive characters (amino acids) of length m from the k-mer, which is lexicographically smallest one in both forward and backward order of the k-mer, where $m < k$ and is fixed.*

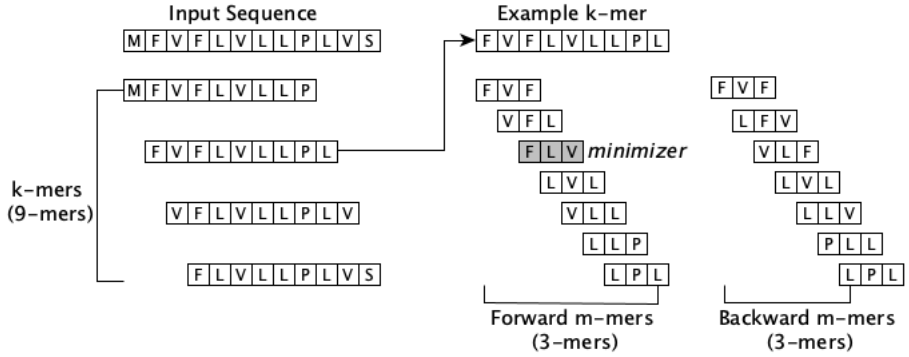


Figure 3.4 Workflow of Minimizer. Firstly, the k-mers (9-mers in this case) are extracted from the sequence. Then for every k-mer, its corresponding minimizer is computed by finding the lexicographically smallest one among the forward and backward m-mers (3-mers in this case).

Algorithm 2 Minimizer Computation

Input: Sequence seq and integer $kSize$ and $mSize$
Output: Set of Minimizers
minimizersList = $\emptyset$
q = [ ] ▷ maintain queue of all m-mers
index = 0 ▷ index of the current minimizer
for $i \leftarrow 1$ to $|seq| - kSize + 1$ **do**
    kmer = $seq[i : i + kSize]$
    if index > 1 **then**
        q.dequeue
        mmer = $seq[i + kSize - mSize : i + kSize]$ ▷ new m-mer
        index $\leftarrow$ index -1 ▷ shift index of current minimizer
        mmer = min(mmer, reverse(mmer)) ▷ lexicographically smallest forw./rever.
        q.enqueue(mmer)
        if mmer $<$ q[index] **then**
            index = $kSize - mSize$ ▷ update minimizer with new m-mer
        end if
    else
        q, index = [ ], 0 ▷ reset the queue
        for $j \leftarrow 1$ to $kSize - mSize + 1$ **do**
            mmer = kmer$[j : j + mSize]$ ▷ compute each m-mer
            mmer = min(mmer, reverse(mmer))
            q.enqueue(mmer)
            if mmer $<$ q[index] **then**

3.2.1.4 Spike2CGR-based Image Encoding

The main idea of Spike2CGR is the same as the minimizer computation as given in Algorithm 2. However, the main difference in the case of Spike2CGR is the preservation of the order of amino acids in m-mers (minimizers). More formally, given a sequence s, we first compute the minimizers using Algorithm 2. Then, we combine all minimizers to make a single sequence s' (see Figure 3.5). Note that s' will be a different sequence from s as it only contains the amino acids within the minimizers. We then repeat the process of computing k-mers (similar to Spike2Vec) from s'. This will give us a list of k-mers computed from s'. We then concatenate all those k-mers to make a new sequence s''. This new sequence s'' is then used as input to the chaos method for image generation (by computing the x and y coordinates of each amino acid in s'' using Equation 3.3 and Equation 3.4 and increment the corresponding pixel values of the amino acids by 1).

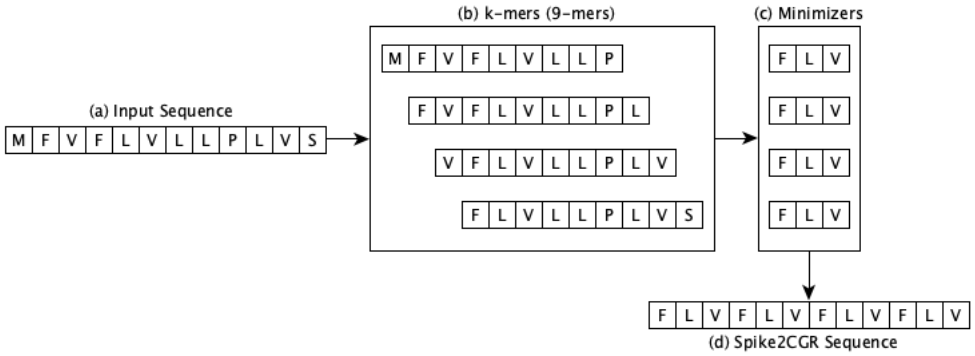


Figure 3.5 Workflow of Spike2CGR for a given sequence. For a given spike sequence, steps from (a) to (d) are followed to generate the corresponding Spike2CGR sequence.

3.2.2 Spaced K-mers & CGR based Image Generation

The generation of an image using the spaced k-mers Singh et al. (2017) of a sequence, rather than using the sequence itself, is investigated in this method. As spaced k-mers provide more meaningful manipulation of amino acids, therefore they enable more information about the sequence to be captured in the corresponding generated image, which can lead to better predictive performance. Spaced k-mers are introduced to overcome the sparsity and high-dimensionality challenges associated with k-mers. For a given peptide sequence, we start by computing its g-mers, and then we use these g-mers to compute the spaced k-mers. Note that $k < g$. For our experiments, we have used $k = 4$ and $g = 9$.

Definition 6. *Spaced k-mers: It is a set of contiguous sub-sequences of length g (where $g < k$) for a given sequence. The algorithmic pseudocode to compute spaced k-mers is given in Algorithm 3.*

Algorithm 3 Spaced k-mers Algorithm

Input: A biological sequence S, k-mer length k, and gap size g
Output: A set of spaced k-mers for the given sequence

```
K = {}                                   ▷ Initialize an empty set of spaced k-mers
for i = 0 to len(S) - (k + g) do
    k-mer = S[i:i+k]
    spaced-k-mer = k-mer[0:g]                                          ▷ where g < k
    K.append(spaced-k-mer)
end for
return set K
```

Our peptide sequence image generation methodologies follow the CGR Jeffrey (1990) concept. We use one existing (P-CGR Löchel et al. (2020)) and 3 different strategies (Static,

Random, RCGR) to do the image encoding of our peptide dataset. The details of each strategy are given below.

3.2.2.1 P-CGR

P-CGR (protein CGR) is proposed to map protein sequences into images by producing n-flakes Tzanov (2015), where n represents the number of amino acids ($n = 20$ for protein sequences). A fractal is constructed by an n-flake following an iterative mechanism from an n-gon.

Definition 7 (n-gon)**.** *An n-gon is a n-sided polygon in geometry.*

For a given peptide sequence, the 2D coordinates (x, y) of its amino acid at index j is obtain by using the following formulas:

$$\begin{aligned} x &= sin(\frac{2\pi j}{n}) \\ y &= cos(\frac{2\pi j}{n}) \end{aligned} \tag{3.5}$$

where n represents the number of amino acids in the peptide sequence. The reason to use the trigonometric functions *sin* and *cos* is to generate the coordinates based on circular mapping instead of linear mapping, which can result in better visualization and clustering of similar amino acids. We use this method as a traditional image-based baseline to perform the evaluation. The algorithmic pseudocode for P-CGR is given in Algorithm 4.

Algorithm 4 P-CGR

Input: Peptide Sequences
Output: 2D Image Representations

for seq in $sequences$ **do**
  $x, y \leftarrow [1], [1]$ ▷ Initialize the starting point
  $point \leftarrow [1, 1]$
  for aa in seq **do** ▷ Loop through the sequence
    $x = sin(\frac{2\pi j}{|seq|})$ ▷ From Equation 3.5
    $y = cos(\frac{2\pi j}{|seq|})$ ▷ From Equation 3.5
    $coord \leftarrow [x, y]$
    $point \leftarrow \frac{point+coord}{2}$
    $x.append(point[0])$
    $y.append(point[1])$
  end for
  $image_{seq} \leftarrow$ GenerateImage(x, y)
end for
return$(image)$

3.2.2.2 Static Chaos Game Representation

In the static Chaos Game Representation (CGR) method, we employ a set of pre-defined axis values for each amino acid of the peptide sequence. This set contains 20 pairs of unique values for the x and y axis corresponding to 20 unique amino acids, as shown in Table 3.1. For a given peptide sequence, the location of its amino acid in the image is determined based on the axis value of that amino acid and the location of the previous amino acid. The algorithmic pseudocode for static CGR is given in Algorithm 5.

3.2.2.3 Random Chaos Game Representation

In this approach, we use a CGR that utilizes a random function (instead of using pre-defined static values) to assign axis values to any amino acid of the peptide sequence. This

Amino Acid	X-Axis Value	Y-Axis Value
A	1	0
C	0.5	0.5
D	0	1
E	0.5	1.5
F	1	2
G	1.5	0
H	1.5	1
I	2	1
K	0	0
L	2	0
M	2	2
N	0.5	0
P	2	0.5
Q	0	0.5
R	0.5	2
S	2	0.5
T	1	1
V	2	1
W	0	2
Y	1	2

Table 3.1 Static Amino acids positions/coordinates for x and y axis in the 2D image.

Algorithm 5 Static Chaos Game Representation

Input: Peptide Sequences
Output: 2D Image Representations
for *seq* in *sequences* **do**
  $x, y \leftarrow [1], [1]$ ▷ Initialize the starting point
  $point \leftarrow [1, 1]$
  for *aa* in *seq* **do** ▷ Loop through the sequence
    $coord \leftarrow \text{AminoAcidCoord}(aa)$ ▷ Table 3.1
    $point \leftarrow \frac{point+coord}{2}$
    $x.append(point[0])$
    $y.append(point[1])$
  end for
  $image_{seq} \leftarrow \text{GenerateImage}(x, y)$
end for
return($image$)

randomly assigned axis value pair (2-D array) is further utilized with the location axis of the previous amino acid to draw the existing amino acid on the image. The algorithmic pseudocode for random CGR is given in Algorithm 6.

Algorithm 6 Random Chaos Game Representation

Input: Peptide Sequences
Output: 2D Image Representations

for *seq* in *sequences* **do**
  $x, y \leftarrow [1], [1]$ ▷ Initialize the starting point
  $point \leftarrow [1, 1]$
  for *aa* in *seq* **do** ▷ Loop through the sequence
    $x = \text{GenerateRandomNumber}()$
    $y = \text{GenerateRandomNumber}()$
    $coord \leftarrow [x, y]$
    $point \leftarrow \frac{point+coord}{2}$
    $x.append(point[0])$
    $y.append(point[1])$
  end for
  $image_{seq} \leftarrow \text{GenerateImage}(x, y)$
end for
return($image$)

3.2.2.4 Rectangular Chaos Game Representation (RCGR)

Our proposed RCGR portrays a similar behavior as P-CGR but it uses the following formulas to get the axis value for an amino acid:

$$\begin{aligned} x &= secant(\frac{2\pi j}{n}) \\ y &= cosecant(\frac{2\pi j}{n}) \end{aligned} \tag{3.6}$$

where n represents the number of amino acids in the peptide sequence. The usage of trigonometric functions secant and cosecant will result in a more rectangular mapping, as they have a strong periodicity in their output. The algorithmic pseudocode for RCGR is given in Algorithm 7.

Furthermore, using all four methods, we have generated the images based on spaced k-mers of the peptide sequences (referred to as S-P-CGR, S-Static, S-Random, and S-RCGR in

Algorithm 7 RCGR

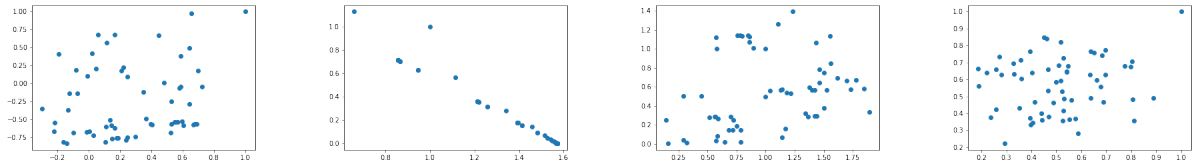
Input: Peptide Sequences
Output: 2D Image Representations

for seq in $sequences$ **do**
  $x, y \leftarrow [1], [1]$ ▷ Initialize the starting point
  $point \leftarrow [1, 1]$
  for aa in seq **do** ▷ Loop through the sequence
    $x = secant(\frac{2\pi j}{|seq|})$ ▷ From Equation 3.6
    $y = cosecant(\frac{2\pi j}{|seq|})$ ▷ From Equation 3.6
    $coord \leftarrow [x, y]$
    $point \leftarrow \frac{point+coord}{2}$
    $x.append(point[0])$
    $y.append(point[1])$
  end for
  $image_{seq} \leftarrow$ GenerateImage(x, y)
end for
return$(image)$

the experiments) and based on the original peptide sequences (referred to as P-CGR, Static, Random, and RCGR in the experiments) to investigate the performance. We referred to P-CGR as the traditional image-based baseline (proposed in Tzanov (2015)). Moreover, an overview of the images generated by various methods for a given peptide sequence is illustrated in Figure 3.6. We can observe that all the generated images differ from each other, which indicates that every method is capturing the sequence information in the image differently. This kind of comparative study can help us to identify the most optimal mapping from sequences to images in terms of predictive performance.

3.2.3 Bézier Curve based Image Generation

Bézier curve Han et al. (2008) is a smooth and continuous parametric curve that is



(a) S-P-CGR (b) S-RCGR (c) S-Static (d) S-Random

Figure 3.6 Graphical representations of different methods using a randomly selected peptide sequence belonging to a *moderately active* category generated by different methods using the spaced k-mers of the sequence.

defined by a set of discrete control points. It is widely used to draw shapes, especially in computer graphics and animation. It has been used in the representation learning domain previously but mainly focusing on extracting numerical features, such as in Hug et al. (2020) which does n-step sequence prediction based on the Bézier curve, Liu et al. (2021) proposed end-to-end text spotting using the Bézier curve, Qiao et al. (2023) does map construction, etc. However, we aim to utilize the Bézier curve to formulate an efficient mechanism for transforming biological sequences into images by effectively mapping the components of a sequence onto a curve. Each component, or character (an amino acid, nucleotide, etc.) of a sequence is represented by multiple lines on the curve which enable more information to be captured in the respective image, hence producing a better representation. The goal of using Bezier curves is to create a visualization that aids in the analysis of protein sequences. This visualization can allow researchers to explore patterns and trends that might provide insights into protein structure and function.

The general formula Baydas & Karakas (2019) of the Bézier curve is

$BZ(t) = \Sigma_{i=0}^{n} \binom{n}{i} t^i (1-t)^{n-i} P_i$

where $0 \leq t \leq 1$, P_i are known as control points and are elements of $\mathbb{R}^k$, and $k \leq n$.

To construct the protein images, we employ a Bézier curve with $n = 3$ and $k = 2$. As images consist of x and y coordinates, therefore $k = 2$ is used. The formulas to determine the coordinates for representing an amino acid in the respective generated image are,

$$x = (1-t)^3 \cdot P_{0_x} + 3 \cdot (1-t)^2 \cdot t \cdot P_{1_x} + 3 \cdot (1-t) \cdot t^2 \cdot P_{2_x} + t^3 \cdot P_{3_x} \tag{3.7}$$

$$y = (1-t)^3 \cdot P_{0_y} + 3 \cdot (1-t)^2 \cdot t \cdot P_{1_y} + 3 \cdot (1-t) \cdot t^2 \cdot P_{2_y} + t^3 \cdot P_{3_y} \tag{3.8}$$

where, (P_{0_x}, P_{0_y}), (P_{1_x}, P_{1_y}), (P_{2_x}, P_{2_y}), & (P_{3_x}, P_{3_y}) denote the x & y coordinates of the four distinct control points respectively.

The algorithm and workflow of creating Bézier-based images are illustrated in Algorithm 8 and Figure 3.7, respectively. We can observe that given a sequence and number of parameters m as input, the algorithm and workflow yield an image as output. Note that m indicates the parameter t shown in the above equations. The process starts by computing the control points by considering the unique amino acids of the given sequence and their respective ASCII values (numerical), as depicted in steps 4-6 of the algorithm and step (b) of the workflow. A control point is made of a pair of numerical values representing the x and y coordinates, where x is assigned the index of the first occurrence of the respective unique amino acid and y holds its ASCII value. Moreover, m linearly spaced random pairs belonging to [0,1] are generated as parameters (mentioned in step 9 and step (c) of the algorithm and workflow respectively). Note that we used $m = 200$ for our experiments. Then the deviation

pair points are generated for every amino acid of the sequence (as exhibited in step 15 of the algorithm and step (d) of the workflow). We utilized 3 deviation pairs to conduct our experiments. After that, modified pair points are obtained by adding the deviation pairs to the corresponding amino acid's control point pair respectively, as shown in step 16 of the algorithm and step (e) of the workflow. Then the Bézier pair points are extracted from the Bézier function by employing (3.7) and (3.8) (as presented in step 19 and step (f) of the algorithm and workflow respectively). Finally, the Bézier pairs are used as x and y coordinates to plot the image (as shown in step 23 and step (g) of the algorithm and workflow respectively). Note that, we get multiple Bézier pairs depending on the value of m and we plot all the pairs in the created image to represent the respective amino acid in the image.

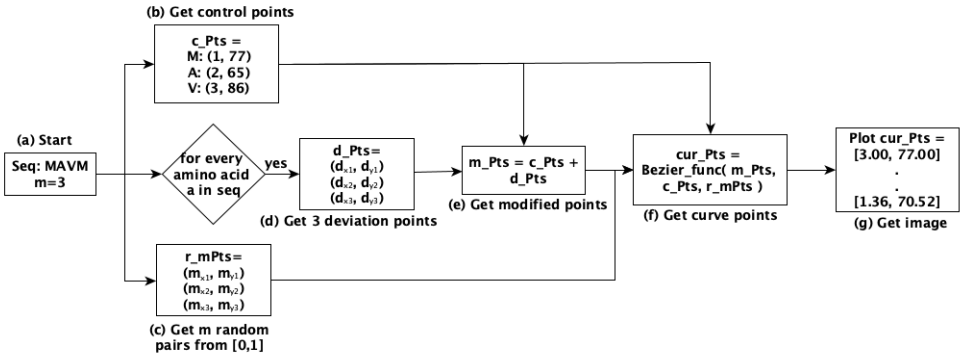


Figure 3.7 The workflow of our system to create an image from a given sequence and a number of parameters m. We have used "MAVM" as an input sequence here. Note that the cur_Pts consists of a set of values for x coordinates and y coordinates.

As Bézier curves are known for their ability to smoothly interpolate control points, using them to connect control points for representing amino acids ensures a visually smooth transition between points, making the visualization more intuitive and easy to interpret. Moreover, introducing randomness to the control points by adding deviations results in con-

Algorithm 8 Bézier Curve Based Image Generation

Input: `Sequence` seq, `No. of Parameters` m
Output: `Image` img

conPoint = {} ▷ dictionary for control points
for $i, aa \in seq$ **do**: ▷ every unique amino acid aa in seq
    conPoint[aa] = $[i, ASCII(aa)]$ ▷ assign control point the index i and ASCII of aa
end for
$xCord$ = [] ▷ list for x coordinates
$yCord$ = [] ▷ list for y coordinates
t_Val = Get m pairs $\in [0, 1]$ ▷ list of m pairs of parameters
$ite = 3$ ▷ no. of deviations pair points. It can have any value.
for $a \in seq$: **do** ▷ every amino acid a in seq
    org_point = conPoint[a] ▷ control point of a
    points = [org_point]
    for $i \in (ite)$: **do**
        dev = Get_Random_Pair ▷ get a random pair
        mod_point = org_point + dev ▷ get a modified control point
        points.append(mod_point)
    end for
    curve_point = Get_Bezier_Point(points, t_Val) ▷ get bezier curve points from bezier func
    $xCord$ = curve_point[:0] ▷ get x coords of curve
    $yCord$ = curve_point[:1] ▷ get y coords of curve
end for
img = plot($xCord$, $yCord$) ▷ get image by plotting x & y coords
return(img)

trolled CGR. While the approach deviates from traditional CGR, it helps reveal patterns that might not be apparent in regular CGR due to the scattering of control points. This randomness mimics the inherent variability and noise present in biological sequences. It can be justified as an attempt to capture the inherent variability in protein sequences that can arise due to mutations, structural differences, or experimental variations.

CHAPTER 4

Experimental Setup

This chapter discusses the experimental details for each of the proposed method. All experiments are conducted using a server having Intel(R) Xeon(R) CPU E7-4850 v4 @ 2.40GHz with Ubuntu 64 bit OS (16.04.7 LTS Xenial Xerus) having 3023 GB memory.

4.1 Dataset Statistics

This section highlights the datasets used for evaluating each of the proposed system.

4.1.1 PSSMFreq2Vec & PSSM2Vec

The PSSMFreq2Vec and PSSM2Vec are evaluated using two datasets, Coronavirus host dataset and SARS-CoV-2 variant dataset.

The Coronavirus host dataset is extracted from both ViPR Pickett et al. (2012); Ali et al. (2022) and GISAID [1]. The statistical detail of this data is given in Table 4.1. Given the spike protein sequence of the Coronavirus, it's corresponding host name is used as label to perform classification using various ML models. Note that for the Coronavirus host data, we also performed sequence alignment on all sequences so that we can run different baseline approaches that only work when all sequences have the same length. We then compare the results for the aligned and unaligned versions of this dataset in the results section.

The SARS-CoV-2 variant dataset is extracted purely from GISAID. It consists of SARS-

[1] `https://www.gisaid.org/`

Host Name	Count	Host Name	Count	Host Name	Count
Bats	153	Bovines	88	Cats	123
Equine	5	Fish	2	Cattle	1
Humans	1813	Pangolins	21	Rats	26
Weasel	994	Birds	374	Turtle	1
Dolphins	7	Environment	1034	Camels	297
Hedgehog	15	Monkey	2	Canis	40
Python	2	Swines	558	Unknown	2
Total	5558				

Table 4.1 The Coronavirus host dataset distribution.

CoV-2 spike sequences along with the information about the corresponding variant. The variant name is used as label for performing classification with the corresponding sequence as input. The statistical distribution of this data is illustrated in Table 4.2. It has 22 unique variants which are the classification labels.

Variant/Lineage	Count	Variant/Lineage	Count	Variant/Lineage	Count
B.1.1.7	976077	B.1.351	20829	B.1.617.2	242820
P.1	56948	B.1.427	17799	AY.4	156038
B.1.2	96253	B.1	78741	B.1.177	72298
B.1.1	44851	B.1.429	38117	AY.12	28845
B.1.160	25579	B.1.526	25142	B.1.1.519	22509
B.1.1.214	17880	B.1.221	13121	B.1.258	13027
B.1.177.21	13019	D.2	12758	B.1.243	12510
R.1	10034				
Total	1995195				

Table 4.2 The SARS-CoV-2 variant dataset distribution.

4.1.2 Hashing2Vec

Hashing2Vec is also validated by employing two datasets. One of them is the same Coronavirus host dataset as discusses above (Table 4.1) used for host classification, while the other consists of SARS-CoV-2 variant data (refer as Spike7k data) but it follows a

different distribution as shown in Table 4.3. This variant data is extracted from GISAID and it is used to perform variant classification. It has 22 unique variants which are the classification labels.

Variant/Lineage	Count	Variant/Lineage	Count	Variant/Lineage	Count
B.1.1.7	3369	B.1.617.2	875	AY.4	593
B.1.2	333	B.1	292	B.1.177	243
P.1	194	B.1.1	163	B.1.429	107
B.1.526	104	AY.12	101	B.1.160	92
B.1.351	81	B.1.427	65	B.1.1.214	64
B.1.1.519	56	D.2	55	B.1.221	52
B.1.177.21	47	B.1.258	46	B.1.243	36
R.1	32				
Total	7000				

Table 4.3 The SARS-CoV-2 variant dataset distribution.

4.1.3 Spike2CGR

This method is yet again validated using two datasets, Coronavirus host data and SARS-CoV-2 variant data. The host data is the same as given in Table 4.1 but since it's image classification, so a more detailed distribution is shown in Table 4.4. Likewise, it employs SARS-CoV-2 variant data but only for 13 unique variants/lineages, and their distribution is given in Table 4.5.

4.1.4 Spaced K-mers & CGR based Image Generation

This approach is validated using the Membranolytic anticancer peptides (ACPs) dataset Grisoni et al. ('2019') contains information about the peptides (protein sequences) and their anti-cancer activity (target labels) on breast cancer cell lines. The target labels are catego-

Host	No. of sequences		
	Training	Validation	Testing
Bat	96	23	34
Bird	242	61	71
Bovine	55	12	21
Camel	186	45	66
Canis	26	5	9
Cat	72	18	33
Cattle	1	0	0
Dolphin	4	1	2
Environment	682	162	190
Equine	3	0	2
Fish	1	0	1
Hedgehog	10	3	2
Human	1159	292	362
Monkey	1	0	1
Pangolin	13	2	6
Python	1	0	1
Rat	22	2	2
Swine	344	104	110
Turtle	1	0	0
Unknown	1	0	1
Weasel	629	160	205
Total	3549	890	1119

Table 4.4 Dataset statistics for different coronavirus infected hosts (5558 in total).

Lineage	Region	Labels	No. Mut. S/Gen.	No. of sequences		
				Training	Validation	Testing
B.1.1.7	UK	Alpha	8/17	9930	2527	3146
B.1.617.2	India	Delta	8/17	1877	450	456
P.2	Brazil	Zeta	3/7	1780	432	533
B.1.429	California	Epsilon	3/5	1079	256	326
P.1	Brazil	Gamma	10/21	994	245	306
B.1.526	New York	Iota	6/16	847	219	255
B.1.351	South Africa	Beta	9/21	837	221	258
B.1.427	California	Epsilon	3/5	835	218	268
B.1.1.529	South Africa	Omicron	34/53	747	178	253
C.37	Peru	Lambda	8/21	732	169	228
B.1.621	Colombia	Mu	9/21	717	168	219
B.1.525	UK and Nigeria	Eta	8/16	714	187	224
P.3	Philippines	Theta	8/17	111	30	34
Total	_	_	_	21200	5300	6238

Table 4.5 Dataset statistics for different coronavirus variants (32738 in total).

rized into “very active”, “moderately active”, “experimental inactive”, and “virtual inactive” groups. This dataset contains 949 peptide sequences distributed among the four categories,

as shown in Table 4.6.

		Peptide Sequence Length			Number of Sequences		
ACPs Category	Count	Min.	Max.	Average	Training	Validation	Testing
Inactive-Virtual	750	8	30	16.64	540	60	150
Moderate Active	98	10	38	18.44	71	7	19
Inactive-Experimental	83	5	38	15.02	61	6	16
Very Active	18	13	28	19.33	14	1	3
Total	949	-	-	-	-	-	-

Table 4.6 ACPs dataset distribution based on their respective activity on the breast cancer cell line. The min, max, and average length of sequences belonging to each category are also mentioned, along with the counts of sequences in test, validation, and train sets for the respective category.

4.1.5 Bézier Curve based Image Generation

We have used 3 distinct protein sequence datasets, a nucleotide-based dataset, a musical dataset, and a SMILES string dataset to evaluate our proposed system. The reason to use such diversified datasets is to show the generalizability of our method for any type of sequence. Each dataset is summarized in Table 4.7.

4.2 Evaluation Metrics

The performance of various models is evaluated using average accuracy, precision, recall, F1 (weighted), F1 (macro), Receiver Operator Characteristic Curve (ROC), Area Under the Curve (AUC), and training run-time metrics. Furthermore, the one-vs-rest approach is used to convert the binary evaluation metrics to multi-class ones. These metrics are reported for all the proposed techniques.

4.3 ML Models

For some proposed mechanism, a set of ML models are used for classifiction-based evaluation. This set belongs to the following list of classifier, Support Vector Machine (SVM), Naive Bayes (NB), Multi-Layer Perceptron (MLP), K-Nearest Neighbors (KNN), Random Forest (RF), Logistic Regression (LR), and Decision Tree (DT).

4.4 DL Models

The image-based encoding methods are validated using DL models to do classification. These DL models are categorized into vision models and tabular models. The vision models employ the image datasets generated from various image-encoding methods. The tabular models are used to classify the tabular data generated from the baseline methods (WDGRL & OHE). These baselines are used to compare the predictive performance of image-based encoding methods.

4.4.1 Vision Models

Various vision models are employed to evaluate the performance of image-based methods through classification. The vision models include custom convolution neural network (CNN), transformer model, and pretrained models. These models are trained by splitting the data into $80-20\%$ train-test sets based on stratified sampling, as it preserves the proportions between the classes. For all the models, the training hyper-parameters used are learning rate

0.003, batch size 64, epochs 10, and optimizer ADAM. The input images are of size 480x480. Furthermore, the negative log-likelihood (NLL) Yao et al. (2019) loss function is employed for training, as it's known to be a cross-entropy loss function for multi-class problems. The details of each model is given below,

4.4.1.1 Custom CNN

We form four types of simple CNN models (1-Layer CNN, 2-Layer CNN, 3-Layer CNN, 4-Layer CNN) by varying the number of "BLOCK" layers. A BLOCK layer consists of a Convolution layer followed by a ReLu activation function and a Max-Pool layer with a kernel size of 5x5 and stride of 2x2. For each model, the BLOCK layers are followed by two fully connected (FC) layers and a final Softmax classification layer. We use these models to observe the impact of an increasing number of layers on the classification performance of our dataset by doing the training from scratch. The architecture of 4-Layer CNN is shown in Figure 4.1.

4.4.1.2 Transformer

A vision transformer model (ViT) is also used by us for performing the classification tasks. As ViT is known to utilize the power of transformer architecture, we want to see its impact on our bio-sequence datasets classifications. In ViT the input image is partitioned into patches, which are then linearly transformed into vectors by a linear embedding module. Note that we used patch size 20 & 8 vector dimensions in our experiments. Then positional embeddings

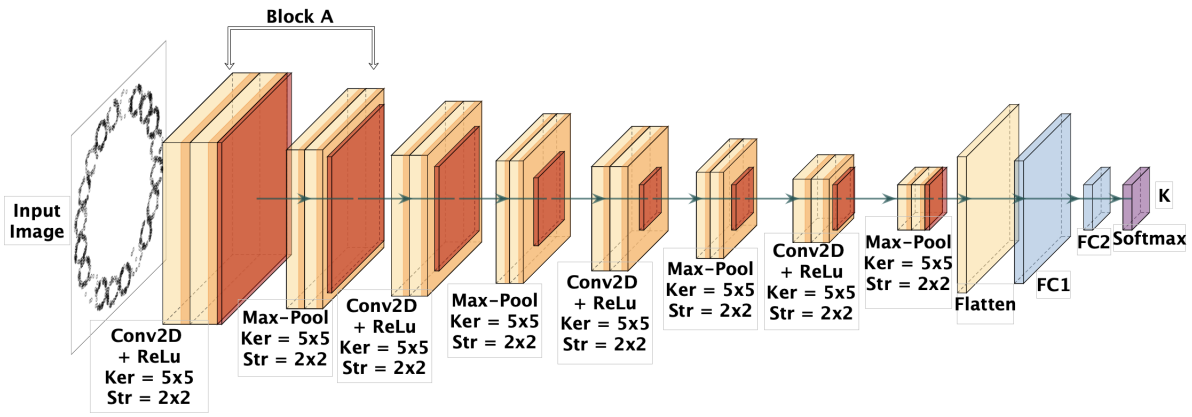


Figure 4.1 The architectures of the 4-layer CNN model, which is used to classify 'K' classes. Here ker represents kernel and str represents stride filter size.

are added to the vectors and they are subsequently processed by two Transformer encoder blocks. Each encoder block consists of a normalization layer, a multi-head self-attention layer with residual connections, a second normalization layer, and a multi-layer perceptron with another residual connection. The final output is directed to a softmax classification module for image label prediction. This design capitalizes on self-attention mechanisms for efficient image classification.

4.4.1.3 Pretrained Models

We also examine the consequences of using pre-trained vision models for classifying our datasets, and for that, we used pre-trained ResNet-50 (He et al. 2016), EfficientNet (Tan & Le 2019), DenseNet (Iandola et al. 2014) and VGG19 (Simonyan & Zisserman 2015) models.

4.4.2 Tabular Models

The feature vectors generated from OHE and WDGRL baselines methods to evalute the image-based encoding methods are classified using two types of tabular models, 3-Layer Tab CNN and 4-Layer Tab CNN. The 3-Layer Tab CNN network contains 3 hidden Linear layers, while 4-Layer Tab CNN has 4 hidden Linear layers. For both, the hidden layers are followed by a final classification Linear layer. For training, these models also follow $80-20\%$ train-test split, with learning rate 0.003, batch size 64, epochs 10, optimizer ADAM, and loss function NLL. The vectors generated by OHE for both SARS-CoV-2 host and variant datasets are of size 27817 for each sequence. However, the vectors computed from WDGRL have 10 size each, as this technique transforms the high dimensional data into low dimension.

4.5 Baselines

Various baseline techniques are utilized to evaluate the proposed algorithms by giving an overview of their comparative performance. The summary of the set of baseline models employed by each proposed method is illustrated in Table 4.8 and the detail of each baseline model is given below.

4.5.1 One Hot Embedding (OHE) Kuzmin et al. (2020)

In this method, a binary feature vector is designed for each alphabet Σ (where Σ contains "ACDEFGHIKLMNPQRSTVWY" characters) in the protein sequence, and the concatenation of all of these represents the sequence. In the binary vector, a 1 is only assigned to the

corresponding character's location while all others have 0 value.

4.5.2 Spike2Vec Ali & Patterson (2021)

For every spike sequence, this method generates sub-strings of length k (k-mers) and creates a frequency vector (Φ) containing the count of each k-mer occurrence in the sequence. Since in our experiments, $k = 3$, the length of the frequency vector for the spike sequence dataset consisting of 20 unique alphabets is $20^3 = 8000$.

4.5.3 PWM2Vec Ali et al. (2022)

PWM2Vec generates embeddings for spike sequences based on the position weight matrix (PWM) concept Stormo et al. (1982). It builds the PWM based on the k-mers of the sequences and uses PWM values as weights for the k-mers to construct the feature vector.

4.5.4 Approximate/String Kernel Farhan et al. (2017)

String kernel works by computing the distance between sequences using the number of matches and mismatches between characters (amino acids) from k-mers and designing the kernel (or gram) matrix. After computing the kernel matrix, classification can be performed by applying kernel PCA (for the non-kernel-based classifiers).

4.5.5 Wasserstein Distance Guided Representation Learning

(WDGRL) Shen et al. (2018) WDGRL is an unsupervised domain adoption technique. It uses the source and target encoded distributions to determine the Wasserstein distance (WD), which is utilized for extracting features from input data with the help of neural networks. It aims to determine the representation by minimizing the estimated WD and optimizing the feature extractor network. It uses a sequence's standard one-hot encoded (OHE) vector as input. The OHE Kuzmin et al. (2020) is an algorithm for creating a fixed-length numerical representation of sequences.

4.5.6 Spaced k-mers Singh et al. (2017)

Feature vectors for sequences based on k-mers frequencies are very large-sized and sparse, and their size and sparsity negatively impact the sequence classification performance. Spaced k-mers introduced the concept of using non-contiguous length k sub-sequences (g-mers) for generating compact feature vectors with reduced sparsity and size. Given a spike sequence as input, it first computed g-mers. From those g-mers, we compute k-mers, where $k < g$. We used $k = 4$ and $g = 9$ to perform the experiments. The size of the gap is determined by $g - k$.

4.5.7 Auto-Encoder Xie et al. (2016)

This approach employs a deep neural network to learn the feature representation of data.

It follows the technique of non-linear mapping from data space X to a lower-dimensional feature space Z, where it iteratively optimizes the objective. It takes the sequences as input. For our experiments, we have used a 2 layered network with an ADAM optimizer and MSE loss function.

4.5.8 SeqVec Heinzinger et al. (2019)

This approach has proposed a way to represent the protein sequences in continuous vectors using the language model named ELMO (Embeddings from Language Models). It captures the biophysical properties from the unlabeled data UniRef50 and creates the embeddings. This process is known as SeqVec (Sequence-to-Vector). It assigns the embeddings to a word by considering the context of a word.

4.5.9 Chaos Löchel et al. (2020)

The FCGR method is referred to as Chaos in this work and it is used as a baseline to evaluate the image-based encoding methods. It produces an n-flakes-based image representation (where n is the number of amino acids), which consists of an image with multiple icosagons. The Chaos method takes a protein sequence as input and yields an image as output by considering the amino acids of the sequence one by one, following Equation 3.3 and Equation 3.4 to get the coordinates of every amino acid in the image.

Dataset	Description
Protein Subcellular Localization	It has 5959 unaligned protein sequences distributed among 11 unique subcellular locations. The associated subcellular location is predicted for a given protein sequence as input.
Coronavirus Host	The unaligned spike protein sequences from various clades of the Coronaviridae family are collected to form this dataset. It contains 5558 spike sequences distributed among 22 unique hosts.
Anticancer Peptides (ACPs)	It consists of 949 unaligned peptide-protein sequences along with their respective anticancer activity on the breast cancer cell lines distributed among the 4 unique target labels.
Human DNA Human DNA (2022)	It consists of $2,000$ unaligned Human DNA nucleotide sequences which are distributed among seven unique gene families. These gene families are used as labels for classification. The gene families are G Protein Coupled, Tyrosine Kinase, Tyrosine Phosphatase, Synthetase, Synthase, Ion Channel, and Transcription Factor containing 215, 299, 127, 347, 319, 94, & 599 instances respectively.
SMILES String Shamay et al. (2018)	It has $6,568$ SMILES strings distributed among ten unique drug subtypes extracted from the DrugBank dataset. We employ the drug subtypes as a label for doing classification. The drug subtypes are Barbiturate [EPC], Amide Local Anesthetic [EPC], Non-Standardized Plant Allergenic Extract [EPC], Sulfonylurea [EPC], Corticosteroid [EPC], Nonsteroidal Anti-inflammatory Drug [EPC], Nucleoside Metabolic Inhibitor [EPC], Nitroimidazole Antimicrobial [EPC], Muscle Relaxant [EPC], and Others with 54, 53, 30, 17, 16, 15, 11, 10, 10, & 6352 instances respectively.
Music Genre Li et al. (2003)	This data has $1,000$ audio sequences belonging to 10 unique music genres, where each genre contains 100 sequences. We perform music genre classification tasks using this dataset. The genres are Blues, Classical, Country, Disco, Hiphop, Jazz, Metal, Pop, Reggae, and Rock.

Table 4.7 The summary of all the datasets used for evaluation.

Proposed Algo.	Baselines
PSSMFreq2Vec & PSSM2Vec	OHE, Spike2Vec, PWM2Vec, String Kernel
Hashing2Vec	Spike2Vec, PWM2Vec, String Kernel, WDGRL, Spaced k-mers, Autoencoder, SeqVec
Spike2CGR	OHE, WDGRL, Chaos
Spaced K-mers & CGR based Images	OHE, WDGRL, Chaos
Bézier Curve based Images	OHE, WDGRL, Chaos, Spike2CGR, RandomCGR

Table 4.8 The list of baseline methods used to evaluate each of the proposed algorithms.

CHAPTER 5

Results & Discussion

This chapter highlights the classification performance achieved by the proposed methods and compares them with their respective baselines. It also compares the data visualization and runtime of embedding generation for the Hashing2Vec algorithm with its corresponding baselines. Furthermore, the statistical analysis and t-SNE Evaluation of PSSMFreq2Vec and PSSM2Vec is also illustrated here.

5.1 Classification Results

The classification tasks are performed by all the proposed methods on their respective datasets for evaluation. The classification results are discussed in detail below.

5.1.1 PSSMFreq2Vec & PSSM2Vec

These algorithms are evaluated using the Coronavirus host dataset and the SARS-CoV-2 variant dataset by performing classification of host and variant respectively, and their performance summary is as follows,

5.1.1.1 Host Data Classification Results

The host-wise classification is performed using aligned and unaligned spike sequences and their respective results are shown in Table 5.1 and Table 5.2. These results summarize the performance obtain by various ML classifiers upon using the embeddings from different

embedding generation methods. From both tables, we can observe that overall PSSM2Vec is outperforming all other embedding techniques. The Spike2Vec and PSSMFreq2Vec methods seem to be comparable in terms of predictive performance. Apart from that, the performance achieved on unaligned data is better or sometimes comparable to) than the aligned data, which indicates that our alignment-free methods are sophisticated enough to extract meaningful information from the spike sequences irrespective of their alignment. They also eradicate the computationally expensive sequence alignment step which could improve our methods' overall runtime and make them more practically applicable in real-world scenarios. Moreover, the training run-time is also optimum for PSSM2Vec, which is because it generates low-dimensional feature vectors.

5.1.1.2 Variant Data Classification Results

The scalability of PSSMFreq2Vec and PSSM2Vec are validated using the SARS-CoV-2 variant data (≈1.9 million sequences) by doing variant-wise classification and the results are demonstrated in Table 5.3. We can observe that PSSMFreq2Vec outperforms all other embedding methods, including PSSM2Vec, in terms of predictive performance. This is an interesting observation here, which indicates that with "Big Data", PSSMFreq2Vec is able to generalize more as compared to PSSM2Vec. Moreover, although PSSM2Vec requires less training time, its performance is lower as compared to PSSMFreq2Vec.

Method	ML. Algo.	Acc.	Prec.	Recall	F1 (Weig.)	ROC AUC	Train Time (Sec.)
OHE Kuzmin et al. (2020)	SVM	0.82	0.83	0.82	0.82	0.83	389.128
	NB	0.67	0.80	0.67	0.65	0.81	56.741
	MLP	0.77	0.76	0.77	0.75	0.71	390.289
	KNN	0.80	0.79	0.80	0.79	0.78	16.211
	RF	0.83	0.83	0.83	0.82	0.83	151.911
	LR	0.83	0.84	0.83	0.82	0.83	48.786
	DT	0.82	0.83	0.82	0.81	0.81	21.581
Spike2Vec Ali & Patterson (2021)	SVM	0.81	0.82	0.81	0.81	0.83	52.384
	NB	0.65	0.77	0.65	0.64	0.74	9.031
	MLP	0.81	0.82	0.81	0.81	0.77	44.982
	KNN	0.80	0.80	0.80	0.79	0.75	2.917
	RF	0.83	0.84	0.83	0.82	0.82	17.252
	LR	0.82	0.84	0.82	0.82	0.83	48.826
	DT	0.81	0.82	0.81	0.81	0.81	4.096
PWM2Vec Ali et al. (2022)	SVM	0.83	0.82	0.83	0.82	0.83	40.55
	NB	0.37	0.68	0.37	0.33	0.69	1.56
	MLP	0.82	0.82	0.82	0.81	0.80	17.28
	KNN	0.82	0.80	0.82	0.81	0.78	2.86
	RF	0.84	0.84	0.84	0.84	0.83	5.44
	LR	0.84	0.84	0.84	0.83	0.83	43.35
	DT	0.82	0.81	0.82	0.81	0.82	3.46
Approx. Kernel Farhan et al. (2017)	SVM	0.78	0.79	0.78	0.77	0.78	16.67
	NB	0.62	0.66	0.62	0.61	0.72	0.19
	MLP	0.79	0.77	0.79	0.77	0.80	8.34
	KNN	0.85	0.84	0.85	0.84	0.80	0.24
	RF	0.82	0.81	0.82	0.81	0.83	1.95
	LR	0.76	0.77	0.76	0.74	0.83	3.80
	DT	0.77	0.77	0.77	0.77	0.82	0.27
PSSMFrq2Vec	SVM	0.83	0.83	0.83	0.82	0.81	50.72
	NB	0.64	0.74	0.64	0.61	0.75	5.90
	MLP	0.83	0.82	0.83	0.83	0.77	33.44
	KNN	0.80	0.80	0.80	0.80	0.75	65.20
	RF	0.84	0.85	0.84	0.83	0.81	11.42
	LR	0.84	0.85	0.84	0.84	0.81	57.55
	DT	0.81	0.82	0.81	0.80	0.79	7.50
PSSM2Vec	SVM	0.78	0.79	0.78	0.76	0.85	1.81
	NB	0.60	0.62	0.60	0.57	0.77	**0.15**
	MLP	0.81	0.81	0.81	0.80	0.89	13.70
	KNN	0.82	0.82	0.82	0.81	0.87	0.66
	RF	**0.86**	**0.86**	**0.86**	**0.85**	**0.91**	1.43
	LR	0.73	0.75	0.73	0.70	0.78	1.91
	DT	0.82	0.82	0.82	0.82	0.89	0.20

Table 5.1 Performance comparison for different embedding methods and different classifiers on the Coronavirus Host (aligned) dataset. Best values are shown in bold.

Method	ML. Algo.	Acc.	Prec.	Recall	F1 (Weig.)	ROC AUC	Train Time (Sec.)
Spike2Vec Ali & Patterson (2021)	SVM	0.84	0.84	0.84	0.83	0.87	45.36
	NB	0.69	0.77	0.69	0.67	0.79	6.02
	MLP	0.81	0.83	0.81	0.81	0.83	46.14
	KNN	0.80	0.81	0.80	0.79	0.79	1.97
	RF	0.84	0.85	0.84	0.84	0.85	10.21
	LR	0.84	0.85	0.84	0.84	0.87	31.00
	DT	0.82	0.83	0.82	0.82	0.85	2.54
Approx. Kernel Farhan et al. (2017)	SVM	0.79	0.80	0.79	0.77	0.78	18.18
	NB	0.60	0.66	0.60	0.57	0.73	0.07
	MLP	0.79	0.78	0.79	0.78	0.75	7.69
	KNN	0.86	0.85	0.86	0.862	0.76	0.21
	RF	0.82	0.82	0.82	0.81	0.78	1.80
	LR	0.76	0.77	0.76	0.74	0.76	2.36
	DT	0.78	0.78	0.78	0.77	0.75	0.24
PSSMFrq2Vec	SVM	0.82	0.82	0.82	0.81	0.84	51.32
	NB	0.67	0.74	0.67	0.64	0.78	6.19
	MLP	0.82	0.84	0.82	0.82	0.81	32.82
	KNN	0.80	0.80	0.80	0.79	0.78	46.85
	RF	0.83	0.83	0.83	0.82	0.84	11.70
	LR	0.84	0.84	0.84	0.83	0.84	33.09
	DT	0.81	0.82	0.81	0.81	0.81	5.79
PSSM2Vec	SVM	0.77	0.78	0.77	0.75	0.86	1.34
	NB	0.68	0.76	0.68	0.65	0.79	**0.14**
	MLP	0.81	0.81	0.81	0.80	0.86	11.72
	KNN	0.82	0.82	0.82	0.81	0.88	0.49
	RF	**0.87**	**0.86**	**0.87**	**0.865**	**0.92**	1.55
	LR	0.72	0.75	0.72	0.70	0.78	1.25
	DT	0.82	0.82	0.82	0.81	0.90	0.19

Table 5.2 Performance comparison for different embedding methods and different classifiers on the Coronavirus Host (unaligned) dataset. Best values are shown in bold.

5.1.2 Hashing2Vec

This method is also evaluated for Coronavirus host data and SARS-CoV-2 variant data (refer to as Spike7k data here).

Method	ML. Algo.	Acc.	Prec.	Recall	F1 (Weig.)	ROC AUC	Train Time (Sec.)
OHE	NB	0.31	0.58	0.31	0.38	0.60	6576.10
	LR	0.57	0.51	0.57	0.50	0.58	191296.4
	RC	0.56	0.49	0.56	0.49	0.57	8725.96
	KC	0.59	0.55	0.59	0.54	0.60	120316.7
Spike2Vec	NB	0.59	0.79	0.59	0.60	0.78	4410.27
	LR	0.88	0.89	0.88	0.87	0.86	140245.19
	RC	0.85	0.83	0.85	0.82	0.82	2985.94
	KC	0.88	0.901	0.88	0.87	0.86	53000.61
PWM2Vec	NB	0.46	0.80	0.46	0.56	0.71	590.13
	LR	0.72	0.71	0.72	0.69	0.72	858.06
	RC	0.70	0.71	0.70	0.67	0.70	138.74
	KC	0.81	0.79	0.81	0.79	0.74	2287.41
PSSMFrq2Vec	NB	0.14	0.73	0.14	0.14	0.71	4605.95
	LR	0.88	0.89	0.88	0.87	0.86	281995.3
	RC	0.86	0.88	0.86	0.84	0.83	7659.69
	KC	**0.89**	**0.905**	**0.89**	**0.88**	**0.87**	90316.71
PSSM2Vec	NB	0.09	0.55	0.09	0.11	0.53	**42.56**
	LR	0.81	0.77	0.81	0.77	0.75	363.13
	RC	0.76	0.70	0.76	0.70	0.64	106.60
	KC	0.82	0.81	0.82	0.81	0.79	695.107

Table 5.3 Variants Classification Results on the SARS-CoV-2 dataset for the top 22 variants (1995195 sequences). Best values are shown in bold.

5.1.2.1 Variant Data Classification Results

The classification performance obtained by Hashing2Vec and its respective baselines is reported in Table 5.4. We can observe that Hashing2Vec is outperforming all other baselines for every evaluation metric, which indicates that it's the optimal embedding generation mechanism in terms of predictive performance. Furthermore, although WDGRL has a minimum train time, it yields the lowest predictive performance. Overall, Hashing2Vec shows optimal classification performance with reasonable training runtime.

5.1.2.2 Host Data Classification Results

The host data classification results are given in Table 5.5. Since the original host classification is done using PWM2Vec in Ali et al. (2022), that is why we are comparing Hashing2Vec

Embedding	Algo.	Acc.	Prec.	Recall	F1 (Weig.)	F1 (Macro)	ROC AUC	Train Time (Sec.)
Spike2Vec Ali & Patterson (2021)	SVM	0.855	0.853	0.855	0.843	0.689	0.843	61.112
	NB	0.476	0.716	0.476	0.535	0.459	0.726	13.292
	MLP	0.803	0.803	0.803	0.797	0.596	0.797	127.066
	KNN	0.812	0.815	0.812	0.805	0.608	0.794	15.970
	RF	0.856	0.854	0.856	0.844	0.683	0.839	21.141
	LR	0.859	0.852	0.859	0.844	0.690	0.842	64.027
	DT	0.849	0.849	0.849	0.839	0.677	0.837	4.286
PWM2Vec Ali et al. (2022)	SVM	0.818	0.820	0.818	0.810	0.606	0.807	22.710
	NB	0.610	0.667	0.610	0.607	0.218	0.631	1.456
	MLP	0.812	0.792	0.812	0.794	0.530	0.770	35.197
	KNN	0.767	0.790	0.767	0.760	0.565	0.773	1.033
	RF	0.824	0.843	0.824	0.813	0.616	0.803	8.290
	LR	0.822	0.813	0.822	0.811	0.605	0.802	471.659
	DT	0.803	0.800	0.803	0.795	0.581	0.791	4.100
String Kernel Farhan et al. (2017)	SVM	0.845	0.833	0.846	0.821	0.631	0.812	7.350
	NB	0.753	0.821	0.755	0.774	0.602	0.825	0.178
	MLP	0.831	0.829	0.838	0.823	0.624	0.818	12.652
	KNN	0.829	0.822	0.827	0.827	0.623	0.791	0.326
	RF	0.847	0.844	0.841	0.835	0.666	0.824	1.464
	LR	0.845	0.843	0.843	0.826	0.628	0.812	1.869
	DT	0.822	0.829	0.824	0.829	0.631	0.826	0.243
WDGRL Shen et al. (2018)	SVM	0.792	0.769	0.792	0.772	0.455	0.736	0.335
	NB	0.724	0.755	0.724	0.726	0.434	0.727	0.018
	MLP	0.799	0.779	0.799	0.784	0.505	0.755	7.348
	KNN	0.800	0.799	0.800	0.792	0.546	0.766	0.094
	RF	0.796	0.793	0.796	0.789	0.560	0.776	0.393
	LR	0.752	0.693	0.752	0.716	0.262	0.648	0.091
	DT	0.790	0.799	0.790	0.788	0.557	0.768	**0.009**
Spaced k-mers Singh et al. (2017)	SVM	0.852	0.841	0.852	0.836	0.678	0.840	2218.347
	NB	0.655	0.742	0.655	0.658	0.481	0.749	267.243
	MLP	0.809	0.810	0.809	0.802	0.608	0.812	2072.029
	KNN	0.821	0.810	0.821	0.805	0.591	0.788	55.140
	RF	0.851	0.842	0.851	0.834	0.665	0.833	646.557
	LR	0.855	0.848	0.855	0.840	0.682	0.840	200.477
	DT	0.853	0.850	0.853	0.841	0.685	0.842	98.089
Auto-Encoder Xie et al. (2016)	SVM	0.699	0.720	0.699	0.678	0.243	0.627	4018.028
	NB	0.490	0.533	0.490	0.481	0.123	0.620	24.6372
	MLP	0.663	0.633	0.663	0.632	0.161	0.589	87.4913
	KNN	0.782	0.791	0.782	0.776	0.535	0.761	24.5597
	RF	0.814	0.803	0.814	0.802	0.593	0.793	46.583
	LR	0.761	0.755	0.761	0.735	0.408	0.705	11769.02
	DT	0.803	0.792	0.803	0.792	0.546	0.779	102.185
SeqVec Heinzinger et al. (2019)	SVM	0.796	0.768	0.796	0.770	0.479	0.747	1.0996
	NB	0.686	0.703	0.686	0.686	0.351	0.694	0.0146
	MLP	0.796	0.771	0.796	0.771	0.510	0.762	13.172
	KNN	0.790	0.787	0.790	0.786	0.561	0.768	0.6463
	RF	0.793	0.788	0.793	0.786	0.557	0.769	1.8241
	LR	0.785	0.763	0.785	0.761	0.459	0.740	1.7535
	DT	0.757	0.756	0.757	0.755	0.521	0.760	0.1308
Hashing2Vec	SVM	0.853	0.858	0.853	0.842	0.685	**0.844**	10.044
	NB	0.598	0.741	0.598	0.637	0.497	0.744	0.375
	MLP	0.759	0.763	0.759	0.752	0.554	0.776	10.972
	KNN	0.825	0.817	0.825	0.811	0.635	0.805	0.557
	RF	0.835	0.842	0.835	0.813	0.642	0.804	4.593
	LR	**0.860**	**0.862**	**0.860**	**0.847**	**0.699**	0.841	17.719
	DT	0.822	0.828	0.822	0.815	0.635	0.812	1.539

Table 5.4 Classification results for different evaluation metrics using the proposed and baseline methods for the Spike7k dataset. Best values are shown in bold.

to PWM2Vec. The results illustrate that the RF method corresponding to Hashing2Vec outperforms PWM2Vec in terms of accuracy, precision, recall, and F1 weighted score, while the SVM of Hashing2Vec has the maximum AUC ROC score. Likewise, Hashing2Vec achieves minimum training time for NB. These results indicate that the Hashing2Vec method has better performance than PWM2Vec.

Embeddings	Algo.	Acc.	Prec.	Recall	F1 (Weig.)	F1 (Macro)	ROC AUC	Train Time (Sec.)
PWM2Vec Ali et al. (2022)	SVM	0.799	0.806	0.799	0.801	0.648	0.859	44.793
	NB	0.381	0.584	0.381	0.358	0.400	0.683	2.494
	MLP	0.782	0.792	0.782	0.778	0.693	0.848	21.191
	KNN	0.786	0.782	0.786	0.779	0.679	0.838	12.933
	RF	0.836	0.839	0.836	0.828	**0.739**	0.862	7.690
	LR	0.809	0.815	0.809	0.800	0.728	0.852	274.917
	DT	0.801	0.802	0.801	0.797	0.633	0.829	4.537
Hashing2Vec	SVM	0.815	0.825	0.815	0.818	0.725	**0.863**	5.591
	NB	0.588	0.649	0.588	0.583	0.585	0.791	**0.146**
	MLP	0.779	0.783	0.779	0.777	0.483	0.735	43.401
	KNN	0.812	0.809	0.812	0.809	0.642	0.817	0.499
	RF	**0.859**	**0.859**	**0.859**	**0.853**	0.735	0.846	5.767
	LR	0.573	0.479	0.573	0.493	0.213	0.591	4.638
	DT	0.800	0.804	0.800	0.800	0.660	0.840	1.855

Table 5.5 Classification results for different evaluation metrics using the proposed and baseline methods for Coronavirus Host dataset. Best values are shown in bold.

5.1.3 Spike2CGR

The four image-based encoding methods (Spike2Vec-based Image Encoding, PWM2Vec-based Image Encoding, Minimizer-based Image Encoding, Spike2CGR-based Image Encoding) are also evaluated using SARS-CoV-2 variant data and Coronavirus host data.

5.1.3.1 Variant Data Classification Results

The variant classification results for the SARS-CoV-2 dataset are reported in Table 5.6. We can observe that the CNN model with 4 layers (using Spike2CGR-based image encoding) outperforms all other methods in terms of average accuracy and recall (best values are shown in bold). For the F1 (macro) score and precision, the 4-layer CNN with PWM2Vec-based image encoding outperforms all other embeddings and DL models. However, for ROC-AUC, the 3-layer CNN model gives the best performance by using minimizer-based image encoding. Moreover, the minimum training runtime is gained by the WDGRL baseline against the 3-layer Tab CNN model. These results illustrate that the performance of the spike to image-

based models is higher than the feature vector-based (baselines) models (OHE and WDGRL), hence indicating that the image representation retains more meaningful information about the input (spike) sequence as compared to the baseline numerical representations for doing classification. We have also reported the performance improvement of the Spike2CGR-based image encoding method as compared to the SOTA Chaos method, and the results show that Spike2CGR-based image encoding has achieved up to 16.7% improvement in terms of ROC-AUC compared with the Chaos method. Overall, Spike2CGR-based image encoding outperforms the Chaos method in most of the evaluation metrics and DL approaches. An important point to note here is that the pre-trained models (RESNET50 and VGG19) are not performing better than the customized CNN models. A possible reason for this behavior is that those models are originally trained on different types of images that have different scales, backgrounds, and foreground information. Hence they fail to generalize on the Chaos-based images. Note that although the performance improvement (for SARS-CoV-2 data) is not very high in some cases compared to Chaos, Spike2CGR can assist doctors, biologists, and relevant government authorities better in taking efficient decisions to minimize the effect and spreading of the coronavirus. Since human health is the most important factor when we talk about fighting a pandemic, a small improvement in the predictive model can help in taking impactful timely decisions.

5.1.3.2 Host Data Classification Results

The host classification results are shown in Table 5.7. Unlike variant classification, it por-

DL Model	Method	Acc. ↑	Prec. ↑	Recall ↑	F1 (Weig.)↑	F1 (Macro)↑	ROC AUC↑	Train Time (hrs.) ↓
3-Tab	OHE	0.472	0.301	0.472	0.368	0.060	0.552	0.594
	WDGRL	0.636	0.457	0.636	0.523	0.263	0.594	**0.380**
4-Tab	OHE	0.637	0.469	0.637	0.528	0.157	0.511	0.977
	WDGRL	0.688	0.517	0.688	0.582	0.227	0.637	0.866
1-Layer CNN	Chaos	0.700	0.680	0.696	0.651	0.563	0.673	8.195
	Spike2Vec	0.733	0.690	0.733	0.679	0.679	0.850	7.779
	PWM2Vec	0.734	0.676	0.734	0.691	0.697	0.844	5.744
	Minimizer	0.743	0.707	0.743	0.709	0.709	0.832	6.171
	Spike2CGR	0.719	0.730	0.766	**0.739**	0.717	0.840	4.992
% improv. of Spike2CGR from SOTA Chaos		1.9	5	7	8.8	15.8	16.7	39.08
2-Layer CNN	Chaos	0.700	0.669	0.697	0.652	0.564	0.645	6.394
	Spike2Vec	0.740	0.730	0.744	0.729	0.736	0.725	7.329
	PWM2Vec	0.740	0.700	0.739	0.688	0.694	0.676	6.615
	Minimizer	0.710	0.710	0.710	0.681	0.581	0.771	6.426
	Spike2CGR	0.633	0.577	0.633	0.559	0.376	0.663	6.193
% improv. of Spike2CGR from SOTA Chaos		-6.7	-9.2	-6.4	-9.3	-18 .8	1.8	3.14
3-Layer CNN	Chaos	0.740	0.722	0.739	0.717	0.696	0.809	5.658
	Spike2Vec	0.750	0.723	0.750	0.715	0.725	0.838	6.919
	PWM2Vec	0.751	0.715	0.751	0.716	0.732	0.846	7.458
	Minimizer	0.750	0.729	0.750	0.721	0.719	**0.851**	6.332
	Spike2CGR	0.770	0.724	0.767	0.734	0.712	0.845	4.758
% improv. of Spike2CGR from SOTA Chaos		3	0.2	2.8	1.7	1.6	3.6	31.23
4-Layer CNN	Chaos	0.740	0.686	0.737	0.706	0.678	0.728	7.986
	Spike2Vec	0.750	0.686	0.749	0.712	0.720	0.842	7.447
	PWM2Vec	0.750	**0.733**	0.745	0.736	**0.747**	0.847	7.720
	Minimizer	0.750	0.726	0.750	0.706	0.709	0.846	7.068
	Spike2CGR	**0.7708**	0.731	**0.768**	0.738	0.714	0.843	10.658
% improv. of Spike2CGR from SOTA Chaos		3	4.5	3.1	3.2	3.6	11.5	-33.45
RESNET50	Chaos	0.680	0.644	0.676	0.641	0.547	0.743	10.654
	Spike2Vec	0.711	0.657	0.710	0.666	0.644	0.759	10.746
	PWM2Vec	0.680	0.589	0.675	0.606	0.507	0.757	10.264
	Minimizer	0.723	0.665	0.723	0.673	0.647	0.802	11.732
	Spike2CGR	0.740	0.661	0.736	0.683	0.626	0.780	14.299
% improv. of Spike2CGR from SOTA Chaos		6	-1.7	6	4.2	7.9	3.7	-34.21
VGG-19	Chaos	0.480	0.233	0.483	0.315	0.050	0.500	27.398
	Spike2Vec	0.470	0.221	0.470	0.301	0.049	0.500	26.599
	PWM2Vec	0.464	0.215	0.464	0.294	0.048	0.500	23.781
	Minimizer	0.480	0.227	0.477	0.308	0.496	0.500	24.459
	Spike2CGR	0.495	0.245	0.495	0.327	0.050	0.500	24.355
% improv. of Spike2CGR from SOTA Chaos		1.5	1.2	1.2	1.2	0	0	8.4

Table 5.6 Variant classification results for SARS-CoV-2 dataset.

trays the best performance for the 1 layer CNN model, and it is because the dataset is small. Hence increasing the model's layers could lead to over-fitting. For the 1 layer CNN model, the Spike2CGR-based image encoding illustrates the best performance in terms of precision, F1 macro, and ROC AUC scores compared to other methods. However, the Minimizer-based image encoding has maximum accuracy, recall, and F1 weighted scores but the difference

between Minimizer-based image encoding and Spike2CGR-based image encoding for these metrics is small. We can also observe that the image-based host classification outperforms the feature vector-based methods (OHE and WDGRL). Furthermore, similar to lineage classification, the pre-trained models (RESNET50 and VGG19) show bad performance for the host classification task. The performance improvement of Spike2CGR-based image encoding compared to Chaos yet again illustrates the performance gain by our method as Spike2CGR gains up to 7.2% improvement in accuracy compared to the Chaos method.

DL Model	Method	Acc. ↑	Prec. ↑	Recall ↑	F1 (Weig.)↑	F1 (Macro)↑	ROC AUC↑	Train Time (hrs.) ↓
3-Tab	OHE	0.625	0.626	0.625	0.566	0.335	0.663	0.032
	WDGRL	0.304	0.137	0.304	0.182	0.041	0.499	**0.029**
4-Tab	OHE	0.613	0.478	0.613	0.534	0.323	0.662	0.067
	WDGRL	0.312	0.130	0.312	0.167	0.035	0.498	0.054
1-Layer CNN	Chaos	0.680	0.707	0.680	0.670	0.517	0.761	0.984
	Spike2Vec	0.728	0.738	0.728	0.711	0.412	0.710	0.738
	PWM2Vec	**0.753**	**0.745**	**0.753**	**0.745**	0.496	0.743	0.950
	Minimizer	0.737	0.735	0.737	0.727	0.514	0.766	1.028
	Spike2CGR	0.743	**0.745**	0.743	0.739	**0.569**	**0.797**	0.711
% improv. of Spike2CGR from SOTA Chaos		6.3	3.8	6.3	5.9	5.3	3.6	27.7
2-Layer CNN	Chaos	0.668	0.684	0.668	0.655	0.410	0.710	1.046
	Spike2Vec	0.735	0.728	0.735	0.713	0.354	0.674	0.821
	PWM2Vec	0.742	0.742	0.742	0.729	0.508	0.7641	0.970
	Minimizer	0.685	0.718	0.685	0.671	0.426	0.706	1.098
	Spike2CGR	0.740	0.734	0.740	0.726	0.428	0.716	0.688
% improv. of Spike2CGR from SOTA Chaos		7.2	5	7.2	7.1	1.8	0.6	34.2
3-Layer CNN	Chaos	0.681	0.677	0.681	0.672	0.470	0.74	0.681
	Spike2Vec	0.718	0.690	0.718	0.695	0.283	0.632	0.717
	PWM2Vec	0.697	0.724	0.697	0.682	0.395	0.689	0.795
	Minimizer	0.731	0.734	0.731	0.719	0.424	0.716	0.960
	Spike2CGR	0.729	0.729	0.729	0.715	0.354	0.677	0.831
% improv. of Spike2CGR from SOTA Chaos		4.8	5.2	4.8	4.3	-11.6	-6.3	-22.02
4-Layer CNN	Chaos	0.624	0.617	0.624	0.606	0.262	0.623	0.991
	Spike2Vec	0.720	0.708	0.720	0.695	0.282	0.630	0.686
	PWM2Vec	0.732	0.720	0.732	0.712	0.294	0.635	0.981
	Minimizer	0.718	0.716	0.718	0.695	0.290	0.636	0.995
	Spike2CGR	0.686	0.668	0.686	0.672	0.283	0.632	0.684
% improv. of Spike2CGR from SOTA Chaos		6.2	5.1	6.2	6.6	2.1	0.9	30.97
RESNET50	Chaos	0.662	0.665	0.662	0.639	0.267	0.621	0.840
	Spike2Vec	0.706	0.672	0.706	0.685	0.277	0.621	0.786
	PWM2Vec	0.663	0.673	0.663	0.663	0.627	0.614	1.020
	Minimizer	0.694	0.671	0.694	0.665	0.266	0.621	1.730
	Spike2CGR	0.691	0.683	0.691	0.663	0.270	0.624	0.786
% improv. of Spike2CGR from SOTA Chaos		2.9	1.8	2.9	2.4	0.3	0.3	6.42
VGG-19	Chaos	0.519	0.475	0.519	0.442	0.158	0.572	3.738
	Spike2Vec	0.506	0.405	0.506	0.407	0.149	0.566	3.390
	PWM2Vec	0.491	0.401	0.491	0.386	0.166	0.579	3.535
	Minimizer	0.509	0.427	0.509	0.427	0.186	0.576	3.969
	Spike2CGR	0.458	0.409	0.458	0.363	0.129	0.559	3.409
% improv. of Spike2CGR from SOTA Chaos		-6.1	-6.6	-6.1	-7.9	-2.9	-1.3	8.80

Table 5.7 Host classification results for coronavirus host dataset.

5.1.4 Spaced K-mers & CGR based Image Generation

The results are shown in Table 5.8. The classification results demonstrate that although the tabular models for OHE fall in the top 5% in terms of average Precision, F1 Macro, F1 Weighted, and ROC AUC score, they tend to lean towards the lower performance bound (even among the top 5%) for most of the evaluation metrics as compared to the image-based methods. Moreover, the image-based methods clearly outperform the WDGRL approach for all the metrics. These results indicate that image representations retain more meaningful information about the peptide sequences than feature-engineering-based methods. We can observe that the traditional image-based baseline, P-CGR, exhibits comparable results in terms of almost all evaluation metrics as compared to our proposed methods for 1-Layer CNN and 2-Layer CNN models. Similarly, our proposed methods (S-P-CGR, S-Static, Static, RCGR) can be seen to yield performance results in the top 5% for almost all the metrics using 1-Layer CNN, 2-Layer CNN models, and 3-Layer CNN models. We can also notice that as the number of layers are increasing in the models, the performance tends to be constant, which indicates that the models are no longer learning. One possible reason for this behavior can be the gradient vanishing issue. With the increased number of layers, the gradient in the back-propagation step can be lost and hence hinders the model's learning ability. Moreover, our dataset size is very small and this also limits the model's learning. Furthermore, the pre-trained vision models are depicting lower performance for most of the metrics as compared to the custom CNN models. This is because those models are trained using different types

of images originally, and they are unable to generalize well to CGR-based images.

5.1.5 Bézier Curve based Images

This section provides an extensive discussion of the classification results obtained by our proposed method and the baseline approaches for 6 distinct classification tasks using 6 different datasets respectively. The details of each one is given below.

5.1.5.1 Protein Subcellular Dataset's Performence

The classification results of the protein subcellular dataset via different evaluation metrics are mentioned in Table 5.9. We can observe that in the case of the custom CNN models, the performance stopped increasing after two layers. It could be because of the dataset being small in size which causes the gradient vanishing problem. Moreover, for the ViT model although the Bézier images have maximum performance as compared to the FCGR and RandomCGR images, however, the overall performance gained by the ViT model is less than the custom CNN models. A reason for this could be the dataset being small in size as ViT typically requires substantial training data to surpass CNN models. Additionally, in ViT a global attention mechanism is used which focuses on the entire image, but in the images generated by all three methods (FCGR, RandomCGR & Bézier) the pertinent information is concentrated in specific pixels, with the remaining areas being empty. Consequently, the global attention mechanism may not be as efficient for these images as a local operation-based CNN model, which is tailored to capture localized features efficiently.

The feature-engineering-based methods are yielding very low performance as compared to our image-based methods (especially FCGR & Bézier) indicating that the image-based representation of bio-sequences is more effective in terms of classification performance over the tabular one. The pre-trained ResNet-50 classifier corresponding to the Bézier method has the optimal predictive performance for all the evaluation metrics. It shows that the ResNet-50 is able to generalize well to the Bézier generated images. It may be due to the architecture of ResNet (like skip connections) enabling the learning on our small dataset. Overall, the pre-trained models (ResNet, VGG19, & EfficientNet) are performing well for the Bézier based images, except the DensetNet model. A reason for DenseNet having very bad performance could be the dataset being small, as DenseNet typically requires large data to yield good performance. Furthermore, among the image-based methods, our Bézier method is tremendously outperforming the baselines for every evaluation metric corresponding to all the vision DL classifiers. This can be because the average length of sequences in the protein subcellular localization dataset is large and our technique uses the Bézier curve to map each amino acid, so a large number of amino acids results in more effective capturing of information about the sequences in their respective constructed images. We have also added results of the Spike2CGR baseline method in Table 5.9 and we can observe that this method is underperforming for all the classifiers for every evaluation metric as compared to our proposed Bézier method. This indicates that the images created by the Bézier technique are of high quality in terms of classification performance as compared to the Spike2CGR-based images. Moreover, the String kernel-based results also showcase very low performance as

compared to the image-based method, hence again indicating that converting sequences to images gives a more effective representation than mapping them to vectors.

5.1.5.2 Coronavirus Host Dataset's Performance

The Coronavirus host dataset-based classification performance via various evaluation metrics is reported in Table 5.10. We can observe that for the custom CNN models, the performance is not directly proportional to the number of hidden layers, i.e., increasing the number of hidden layers does not result in better performance, as most of the top values reside corresponding to the 1-layer CNN model and the 2-layer CNN model. This could be because the host dataset is not large enough to tackle a heavy CNN model, hence ending up having a gradient vanishing problem, which stops the model from learning. Apart from that, the ViT model is exhibiting lower performance than the custom CNN model and it can be yet again due to the dataset being small. Moreover, among the pre-trained models, ResNet-50 & VGG19 are showcasing nearly similar performance as the custom CNN classifiers (with Bézier-based images yielding maximum performance), which indicates that these models are able to generalize well using the images created by our Bézier method. However, DenseNet and EfficientNet are demonstrating very low performance for all evaluation metrics may be because the size of host data is small and these models typically need large data to attain good performance. Additionally, the feature-engineering-based methods lean towards a lower performance bound for all the evaluation metrics corresponding to both 3-layer Tab CNN & 4-layer Tab CNN, and most of the ML classifiers based on the String kernel also

showcase less performance. This indicates that converting the host sequences into images can preserve more relevant information in the respective images about the sequence in terms of classification performance as compared to converting them into vectors. Furthermore, among the image generation methods, RandomCGR has the lowest performance for every metric while Bézier (our method), Spike2CGR, and FCGR have comparable performance as they yield most of the top values for all the metrics. Overall, Bézier seems to perform well for the host classification task, implying that the images generated by it are of good quality for classification.

5.1.5.3 ACP Dataset's Performance

The classification performance achieved using the ACP dataset for various evaluation metrics is summarized in Table 5.11. We can observe that increasing the number of inner layers for the custom CNN models does not enhance the predictive performance, as 1-layer CNN & 2-layer CNN models portray higher performance. This could be because the ACP dataset is very small, so using a large model can cause a gradient vanishing challenge and, hence, hinder the learning process. Additionally, the ViT model is yielding lower performance than the custom CNN models and it can be due to yet again the dataset being very small. Moreover, the pre-trained ResNet-50 and VGG19 models depict very similar performance as the custom CNN models. This shows that the ResNet and VGG19 models are able to generalize well to our Bézier-based data. However, the EfficeintNet and Denset classifiers portray very low performance for every evaluation metric. It can be due to their architectures

which require large data for fine-tuning the model, however, our dataset is extremely small. Furthermore, the feature-engineering-based embedding approaches are overall showcasing bad performance (except for 4 tab CNN OHE) as compared to the image-based methods. It implies that the bio-sequences's information is effectively preserved in the respective image form rather than the vector form generated from the feature-engineering methods in terms of predictive performance. Note that, although the String kernel embedding-based ML classifiers are yielding the highest performances corresponding to every evaluation metric, our method's performance is also close to it, which means that our method is also yielding an effective representation for sequences. For the image-based embedding methods, we can notice that our method (Bézier) and the FCGR baselines illustrate comparable predictive results, while RandomCGR and Spike2CGR lean toward the lower performance bound. Overall, we can claim that the Bézier method exhibits good performance for the ACP classification task.

5.1.5.4 Human DNA Dataset Performance

The classification results for the DL model using the Human DNA dataset are given in Table 5.12. We can observe that the pre-trained vision models and the vision transformer classifier are yielding very low performance corresponding to every image-based strategy. It can be again due to the gradient vanishing problem because of the small size of the dataset. Moreover, the customer CNN models are obtaining high performance, especially for the 1-layer CNN model and 2-layer CNN model. Note that increasing the number of layers in the custom CNN models is reducing the performance, and a small dataset could be a

reason for this behavior too. We can also notice that our proposed Bézier method is able to achieve performance in the top 5% for almost every evaluation metric corresponding to the custom CNN classifiers. Furthermore, the image-based methods clearly outperform the feature-engineering ones, hence indicating that converting the nucleotide sequences to images can retain more information about the sequences as compared to mapping them to vectors in terms of classification predictive performance. Similarly, the String kernel method-based ML classifiers, except RF, also portray less performance than the custom CNN models which yet again proves that converting sequences into images is more effective than mapping them to vectors.

5.1.5.5 SMILES String Dataset Performance

The classification results for the DL model using the SMILES String dataset are given in Table 5.13. We can observe that, the performance achieved by all the classifiers corresponding to every embedding strategy (image or vector) is very good and similar to each other, except for the DenseNet and EfficientNet models which have bad results. A reason for the bad results could be the small size of the data as DenseNet and EfficientNet usually operate on large datasets to have optimal performance. Note that, although most of the classifiers portray similar results, our method achieves the maximum performance. Moreover, as this data contains sequences constituted of more than 20 unique characters, therefore, the FCGR & Spike2CGR methods failed to operate on them. Furthermore, our image-based method is performing better than the tabular ones (feature-engineering-based and String kernel-based),

hence obtaining images of sequences is more useful for the classification tasks.

5.1.5.6 Music Genre Dataset Performance

The classification results for the DL model using the Music Genre dataset are given in Table 5.14. An important point to note here is that since the number of unique characters in the music data is > 20, the traditional FCGR and Spike2CGR methods fail to run on such datasets. In general, although the RandomCGR method performs better using classical vision models, the performance drastically reduces compared to the proposed method on the pre-trained vision models (e.g. see results for VGG-19 results in Table 5.14). Such behavior supports our argument that in general, the proposed method improves the performance of the pre-trained models in terms of sequence classification. Moreover, the image-based methods are clearly outperforming the feature-engineering and String-kernel baselines, hence image representations are more promising for doing classification than the tabular ones.

5.2 Statistical Analysis of Feature-Engineering-based Encoding Methods

The feature-engineering-based encoding methods are further evaluated using statistical analysis and their details are as follows,

5.2.1 Data Visualization (Hashing2Vec)

The t-distributed stochastic neighbor embedding (t-SNE) Van der Maaten & Hinton (2008) is utilized to identify any hidden patterns in the Spike7k data. This method works by mapping the high dimensional input data into 2D space but preserves the pairwise distance

between data points in high dimensions. This visualization aims to highlight if different embedding methods introduce any changes to the overall distribution of data. For the Hashing2Vec embedding method along with its respective baselines, Figure 5.1 illustrated the t-SNE-based visualization (with variants as labels as shown in legends) of the Spike7k dataset. We can observe that overall the B.1.1.7 (Alpha) variant forms a single huge group (shown in yellow color) since its representation in the dataset is larger than other variants. Moreover, we can observe that Hashing2Vec can preserve the structure of the data similar to Spike2Vec, PWM2Vec, Spaced k-mers, and String kernel. The WDGRL shows a scattered t-SNE plot, which means that the overall structure of data, in that case, is disturbed, hence the performance of the embeddings will not be as good compared to other embedding methods (this behavior is also observed in classification results in the next section, in Table 5.4).

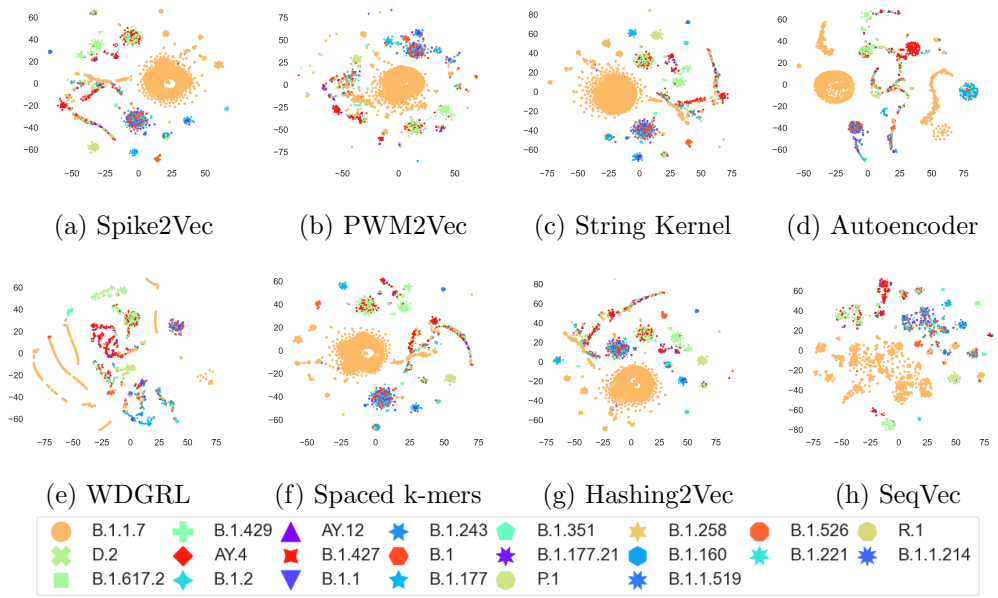


Figure 5.1 t-SNE plots using different embeddings for 7000 Spike7k dataset sequences. This figure is best seen in color.

5.2.2 Embedding Generation Run-Time (Hashing2Vec)

To evaluate the embedding generation computation time for Hashing2Vec and its respective baselines, we report the runtime in Table 5.15 for the Spike7k dataset. We can observe that Hashing2Vec takes the lowest time to generate the feature vectors as compared to the baseline methods. The PWM2Vec is the second-best while the SeqVec takes the most time for feature vector generation. We observed the same behavior regarding the embedding generation runtime in the case of coronavirus host data as well. We also provide % improvement for Hashing2Vec from PWM2Vec (second best in terms of runtime) and SeqVec(worst in terms of runtime) using the following expression for the Spike7k dataset:

$$\% \text{ improvement} \quad = \quad \frac{R_{Baseline} - R_{Hashing2Vec}}{R_{Baseline}} \times 100 \tag{5.1}$$

where $R_{Baseline}$ represents the runtime of baselines PWM2Vec and SeqVec embedding methods while $R_{Hashing2Vec}$ corresponds to the run-time for Hashing2Vec embedding computation. We can observe that Hashing2Vec improves the runtime performance by 68.7% and 99.8% as compared to PWM2Vec and SeqVec, respectively. The runtime for computing PWM2Vec and Hashing2Vec with the increasing number of sequences is shown in Figure 5.2 for the Spike7k dataset. We can see that Hashing2Vec significantly outperforms the PWM2Vec (fastest among other embeddings) in terms of runtime with any number of sequences. Additionally, we can observe that the increasing runtime trend for Hashing2Vec is very slow as compared to the PWM2Vec, which makes it more suitable for Big Data. Overall, we can

observe that the proposed embedding, called Hashing2Vec not only performs slightly better in terms of predictive performance as compared to the baselines, but it also preserves the overall structure of the data similar to the recently proposed embedding methods. Moreover, Hashing2Vec can be generated very quickly as compared to the other methods, making it an ideal choice while dealing with larger-sized datasets because of its scalability property.

Embeddings	Runtime (Seconds)
Spike2Vec Ali & Patterson (2021)	354.061
PWM2Vec Ali et al. (2022)	163.257
String Approx. Farhan et al. (2017)	2292.245
WDGRL Shen et al. (2018)	438.188
Spaced k-mers Singh et al. (2017)	12901.808
Auto-Encoder Xie et al. (2016)	181.70052
SeqVec Heinzinger et al. (2019)	32500.19
Hashing2Vec (ours)	**51.094**
% Improv. of Hashing2Vec from PWM2Vec	68.7%
% Improv. of Hashing2Vec from SeqVec	99.8%

Table 5.15 Embedding generation runtime for different methods using the Spike7k dataset. Best value is shown in bold. The percentage improvement of runtime (see Equation (5.1)) is also given for Hashing2Vec.

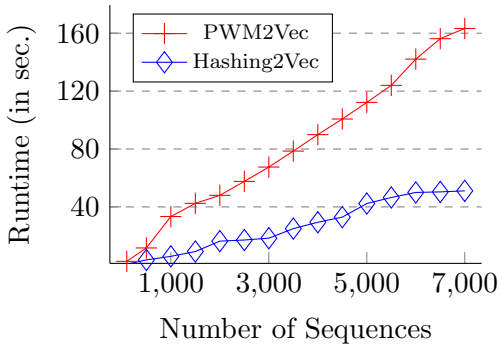


Figure 5.2 Runtime for embedding generation of PWM2Vec and Hashing2Vec with increasing number of sequences for the Spike7k dataset. The figure is best seen in color.

5.2.3 Compactness Analysis & t-SNE Evaluation (PSSMFreq2Vec & PSSM2Vec)

We use the t-distributed stochastic neighbor embedding (t-SNE) approach Van der Maaten & Hinton (2008) to evaluate the (hidden) patterns in the data. To see how well the data is preserved by t-SNE in 2 dimensions given different feature embeddings as input, we use *k-ary neighborhood agreement* (k-ANA) method Zhu & Ting (2021). The k-ANA test computes the nearest neighbors in the original High Dimensional (HD) data and the Low Dimensional

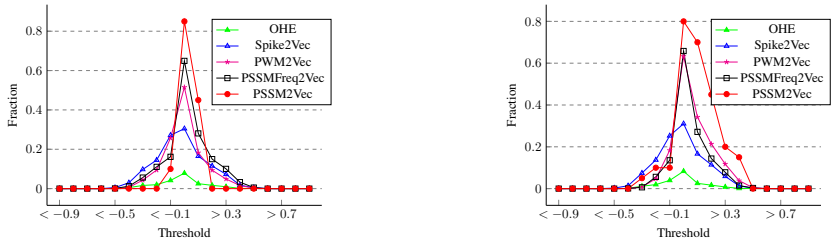
data (LD) (from t-SNE). It then takes the intersection of these to evaluate the number of neighbors on which both HD and LD agree. Using this intersection, a value $R(k)$ is calculated for evaluating the preservation of k-ary neighborhoods, and $R(k)$ is defined as follows: $R(k) = \frac{(n-1)Q(k)-k}{n-1-k}$, where $Q(k) = \Sigma_i^n \frac{1}{nk} |kNN(x_i) \bigcap kNN(x_i')|$. Similarly, $kNN(x)$ represents the set of nearest neighbors of x in high dimensions (HD). Moreover, $kNN(x')$ represents the nearest neighbors in low dimensions (LD). The $R(k) \in [0, 1]$ and higher value of $R(k)$ indicates better preservation of the neighborhood in LD space. For our experiment, we have aggregated the $R(k)$ values for k from 1 to 99 and calculated the area under the curve formed by $R(k)$ using the following expression: $AUC_{RNX} = \frac{\Sigma_k \frac{R(k)}{k}}{\Sigma_k \frac{1}{k}}$. The values of AUC_{RNX} for the baselines and proposed model are given in Table 5.16. We can observe that PSSM2Vec performs better than the other models for the Coronavirus Host dataset. This means that t-SNE is able to preserve the distance (hence the global structure of the data) between sequences more accurately using PSSM2Vec. For the SARS-CoV-2 data, PWM2Vec performs better than the other models. However, PSSM2Vec also performs comparably to PWM2Vec. Since PWM2Vec does not work with unaligned sequences, using PSSM2Vec becomes even more relevant as it works for both aligned and unaligned sequences. In general, since PSSM2Vec is better in terms of predictive performance, training runtime, and preserving the overall structure of data, we believe that it is more applicable in real-world scenarios for the classification of biological sequences.

Furthermore, to get a better idea about the overall time taken by each method to generate embeddings, we reported the feature vector computation runtime for different embedding

methods in Table 5.17. We can again observe that PSSM2Vec takes very little time to generate in the case of both datasets. On the SARS-CoV-2 dataset, while other embedding methods took more than 3 days, PSSM2Vec just took 4.25 hours to generate feature vectors for 1.9 million sequences.

One way to evaluate the effectiveness of the feature embeddings is to analyze their compactness. For this purpose, we perform statistical analysis, including Pearson and Spearman Correlation. We compute the correlation values for different features of embeddings (corresponding to class labels) and report the fraction of attributes in each feature embedding having a high correlation corresponding to the class labels. The Pearson correlation values for different thresholds are reported in Figure 5.3a (for the Coronavirus Host dataset). Similarly, the Spearman correlation values for different thresholds (ranging from -1 to 1) and embeddings are reported in Figure 5.3b (for the Coronavirus Host dataset). We can observe that, overall, PSSM2Vec and PSSMFreq2Vec have the highest fraction of values for both Pearson correlation and Spearman correlation. This shows that more features in PSSM2Vec are highly correlated with the class label, demonstrating that these embeddings are the most compact. This compactness is also highlighted in terms of obtaining better predictive performance in terms of sequence classification (as given in Table 5.1 "for aligned data", Table 5.2 "for unaligned data" and Table 5.3 "for SARS-CoV-2 data"). In summary, we have the following properties of the PSSM2Vec: (i) it is better (or sometimes comparable) in terms of predictive performance, training runtime, and embedding generation runtime, (ii) it easily scales to millions of sequences, (iii) it works for both aligned and unaligned sequences, and

(iv) it is better in terms of feature vector compactness (computed using Pearson and Spearman correlation). Therefore, we can conclude that in a real-world scenario, using PSSM2Vec is more appropriate than other embedding approaches for biological sequence classification.



(a) Pearson Correlation (b) Spearman Correlation

Figure 5.3 Correlation values for Coronavirus Host data. (a) and (b) show the fraction of features having correlation values greater than or less than the thresholds (on the x-axis). The fractions are computed by taking the denominator as the size of embeddings (69960 for OHE, 8000 for Spike2Vec, 3490 for PWM2Vec, 8000 for PSSMFreq2Vec, and 60 for PSSM2Vec).

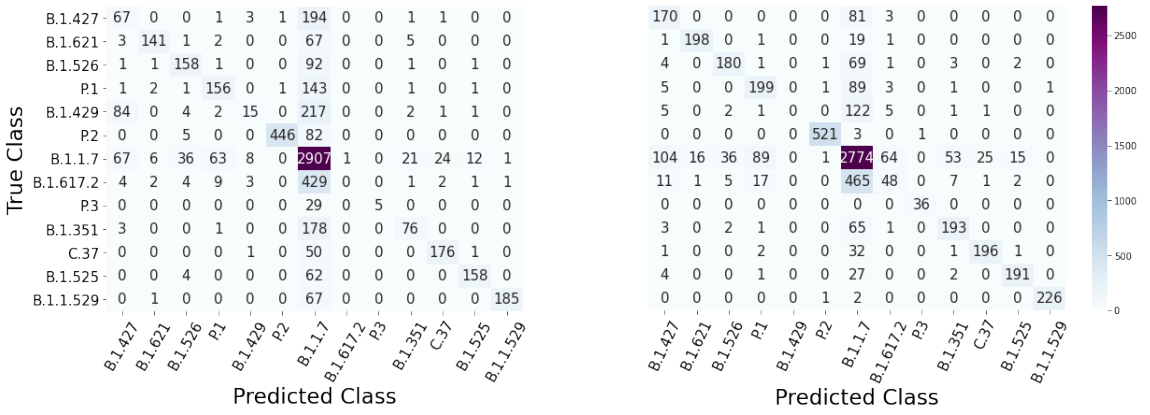
5.3 Statistical Analysis of Spike2CGR

The image-based-based encoding methods are further evaluated using the confusion matrices and their details are given below. The reasons for different image-based encoding methods yielding different images is also highlighted in this section.

5.3.1 Confusion Matrices

We also investigated the confusion matrices of the best performing model (4-layer CNN) for Chaos (the SOTA approach Löchel et al. (2020)) and Spike2CGR-based embedding (see Figure 5.4) for variant classification. We can observe that although for the Alpha variant (B.1.1.7), Chaos-based embedding has the highest true positive count. However, for all other

classes, Spike2CGR-based embedding performs better. This behavior shows that Chaos tends to focus more on the label with high frequency in the data (a typical class imbalance problem) since the Alpha variant has the highest count in the dataset. Likewise, the confusion matrices for host classification using Chaos and Spike2CGR embeddings for the best performing model (1 layer CNN) are given in Figure 5.5. They also portray a similar behavior where Spike2CGR is performing better for most of the hosts as compared to Chaos.

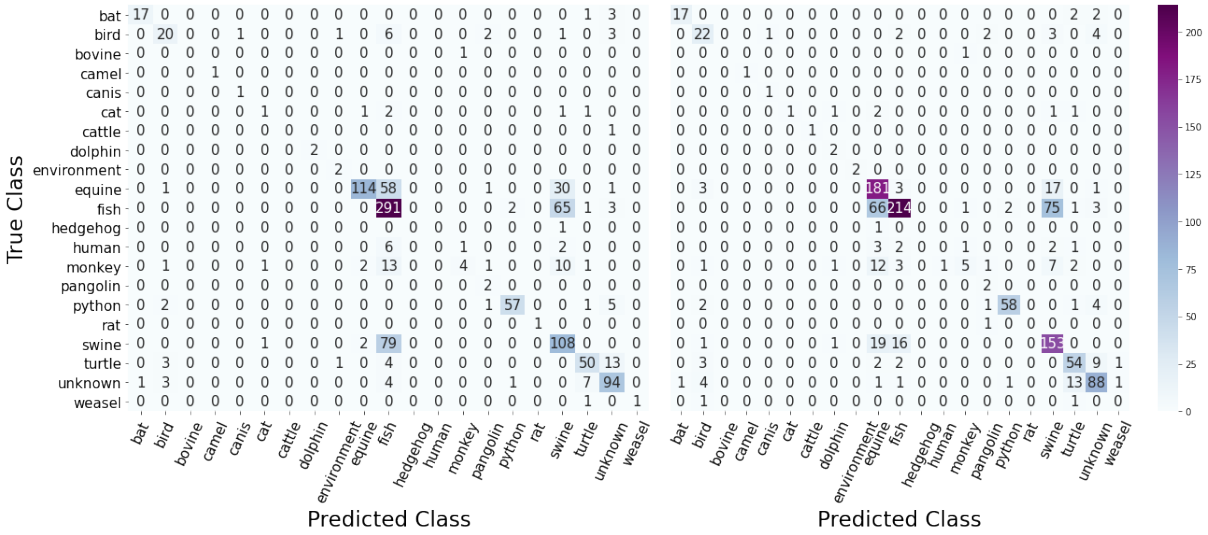


(a) Chaos (b) Spike2CGR

Figure 5.4 Confusion matrices comparison of Chaos and Spike2CGR based encoding to perform variant classification using the best performing model (4-layers CNN).

5.3.2 Reasons of Different Image-based Embeddings Generating Different Images

After the analysis of classification results, a natural question arises why results for different embedding methods are not similar? A few fundamental facts for this behavior are the following:



(a) Chaos (b) Spike2CGR

Figure 5.5 Confusion matrices comparison of Chaos and Spike2CGR based encoding to perform host classification using the best performing model (1-layers CNN).

1. The i^{th} amino acid (pixel) drawn using the CGR for a given sequence corresponds to a specific position in the spike sequence (and it holds some local meaning). Therefore, no other sub-sequence within that spike sequence may have similar information/pattern (up to the resolution of the screen). Hence, there is a one-to-one mapping between the sub-sequence patterns of a given spike sequence and pixels of the CGR. If there is a mutation in the sequence, that mutation should be highlighted clearly in the resultant visual representation using CGR.

2. If we manipulate the sequence (in a biologically meaningful way) using different embedding tricks such as minimizers, the amino acid positions should be disturbed and will no longer remain similar to the original spike sequence. This disturbance of the sequence may affect the performance of classifiers positively or negatively.

3. Because of this manipulation of the protein sequence, the "*area of interest*" within the CGR image could be different for different embeddings.

4. Assigning a certain weight to the pixels (as done using PWM2Vec) is also an important factor for classification as it could affect the resolution of the image.

5.4 Overall Results Summary

The overall summary of the results discussed above in detail is as follows,

1. The PSSM2Vec and PSSMFreq2Vec are shown to be alignment-free and compact feature-engineering methods with the ability to be scalable. They also demonstrate high predictive performance using various ML methods, hence these embeddings are effective and efficient. They have also shown to retain the original data structure in the low dimensional space.

2. The Hashing2Vec algorithms is an alignment-free and fast feature-engineering method. It is fast in terms of embedding generation time. It achieves high classification performance using ML models and shown to identify the hidden data patterns.

3. The image-based encoding methods (Spike2CGR, Spaced K-mers & CGR based Image Generation, Bézier Curve based Images) are alignment-free. They have enabled the application of sophisticated DL classification models on the sequence data and illustrated high predictive performance using these models.

DL Model	Method	Acc.	Prec.	Recall	F1 (Weig.)	F1 (Macro)	ROC AUC	Train Time (hrs.)
3-Tab	OHE	0.768	**0.839**	0.768	0.790	**0.452**	**0.719**	0.042
	WDGRL	0.615	0.740	0.615	0.660	0.326	0.603	**0.0001**
4-Tab	OHE	0.796	**0.843**	0.796	**0.807**	**0.474**	**0.736**	0.056
	WDGRL	0.631	0.754	0.631	0.673	0.346	0.623	**0.0002**
1-Layer CNN	P-CGR	**0.863**	**0.831**	**0.863**	**0.844**	**0.490**	0.677	0.357
	Static	0.849	**0.820**	**0.849**	**0.825**	**0.467**	0.657	0.271
	Random	0.792	0.638	0.792	0.707	0.221	0.497	0.404
	RCGR	0.796	0.756	0.796	0.773	0.385	0.598	0.342
	S-P-CGR	**0.842**	**0.810**	**0.842**	**0.819**	0.423	0.637	0.412
	S-Static	**0.835**	**0.795**	**0.835**	**0.809**	0.409	0.627	0.312
	S-Random	0.701	0.579	0.701	0.599	0.241	0.511	1.385
	S-RCGR	0.803	0.737	0.803	0.756	0.347	0.563	0.468
2-Layer CNN	P-CGR	**0.852**	**0.833**	**0.852**	**0.837**	**0.489**	0.676	0.419
	Static	**0.821**	0.710	**0.821**	0.759	0.318	0.566	0.536
	Random	0.800	0.640	0.800	0.711	0.222	0.500	0.389
	RCGR	**0.821**	**0.809**	**0.821**	0.775	0.372	0.584	0.332
	S-P-CGR	**0.863**	**0.835**	**0.863**	**0.842**	**0.467**	0.666	0.430
	S-Static	**0.856**	**0.827**	**0.856**	**0.836**	**0.463**	0.662	0.385
	S-Random	0.800	0.640	0.800	0.711	0.222	0.500	0.416
	S-RCGR	0.796	0.727	0.796	0.751	0.337	0.556	0.406
3-Layer CNN	P-CGR	0.800	0.640	0.800	0.711	0.222	0.500	0.490
	Static	**0.835**	**0.853**	**0.835**	**0.810**	0.391	0.651	0.557
	Random	0.800	0.640	0.800	0.711	0.222	0.500	0.391
	RCGR	**0.821**	**0.807**	**0.821**	0.778	0.376	0.583	0.529
	S-P-CGR	**0.838**	**0.821**	**0.838**	**0.806**	0.370	0.657	0.462
	S-Static	0.800	0.640	0.800	0.711	0.222	0.500	0.392
	S-Random	0.800	0.640	0.800	0.711	0.222	0.500	0.436
	S-RCGR	0.807	0.690	0.807	0.730	0.254	0.518	0.389
4-Layer CNN	P-CGR	**0.831**	0.735	**0.831**	0.779	0.329	0.586	0.498
	Static	0.800	0.640	0.800	0.711	0.222	0.500	0.512
	Random	0.800	0.640	0.800	0.711	0.222	0.500	0.435
	RCGR	0.800	0.640	0.800	0.711	0.222	0.500	0.536
	S-P-CGR	0.800	0.640	0.800	0.711	0.222	0.500	0.563
	S-Static	0.800	0.640	0.800	0.711	0.222	0.500	0.474
	S-Random	0.800	0.640	0.800	0.711	0.222	0.500	0.460
	S-RCGR	0.800	0.640	0.800	0.711	0.222	0.500	0.506
RESNET50 Pre-Trained Model	P-CGR	0.800	0.642	0.800	0.712	0.222	0.501	1.317
	Static	0.800	0.640	0.800	0.711	0.222	0.500	1.159
	Random	0.800	0.640	0.800	0.711	0.222	0.500	1.387
	RCGR	0.800	0.640	0.800	0.711	0.222	0.500	1.374
	S-P-CGR	**0.828**	**0.801**	**0.828**	0.791	0.357	0.633	1.043
	S-Static	**0.828**	0.768	**0.828**	0.783	0.369	0.585	1.273
	S-Random	0.800	0.640	0.800	0.711	0.222	0.500	1.170
	S-RCGR	0.800	0.640	0.800	0.711	0.222	0.500	1.181
VGG-19 Pre-Trained Model	P-CGR	0.803	0.684	0.803	0.720	0.243	0.509	1.189
	Static	**0.824**	0.713	**0.824**	0.761	0.323	0.565	1.153
	Random	0.800	0.640	0.800	0.711	0.222	0.500	1.054
	RCGR	0.800	0.640	0.800	0.711	0.222	0.500	1.06
	S-P-CGR	**0.817**	0.713	0.817	0.761	0.318	0.576	1.185
	S-Static	**0.828**	0.737	**0.828**	0.779	0.353	0.616	1.573
	S-Random	0.800	0.640	0.800	0.711	0.222	0.500	1.377
	S-RCGR	0.800	0.640	0.800	0.711	0.222	0.500	1.430

Table 5.8 Classification results for different models and algorithms for ACPs dataset. The top 5% best values for each evaluation metric are shown in bold.

Category	DL Model	Method	Acc.	Prec.	Recall	F1 (Weig.)	F1 (Macro)	ROC AUC	Train Time (hrs.)
Tabular Models	3-Tab	OHE	0.449	0.405	0.449	0.401	0.227	0.667	0.398
		WDGRL	0.458	0.315	0.458	0.354	0.163	0.751	0.109
	4-Tab	OHE	0.404	0.409	0.404	0.384	0.215	0.657	0.525
		WDGRL	0.457	0.309	0.457	0.351	0.161	0.708	0.130
String Kernel	-	SVM	0.496	0.510	0.496	0.501	0.395	0.674	5.277
	-	NB	0.301	0.322	0.301	0.265	0.243	0.593	0.136
	-	MLP	0.389	0.390	0.389	0.388	0.246	0.591	7.263
	-	KNN	0.372	0.475	0.372	0.370	0.272	0.586	0.395
	-	RF	0.473	0.497	0.473	0.411	0.218	0.585	7.170
	-	LR	0.528	0.525	0.528	0.525	0.415	0.678	8.194
	-	DT	0.328	0.335	0.328	0.331	0.207	0.568	2.250
Custom CNN Models	1-Layer	FCGR	0.545	0.542	0.545	0.527	0.386	0.653	3.065
		RandmCGR	0.292	0.172	0.292	0.211	0.102	0.528	6.443
		Spike2CGR	0.460	0.453	0.460	0.432	0.277	0.603	6.879
		Bézier	0.948	0.919	0.948	0.931	0.769	0.890	3.455
	% impro. of Bézier from FCGR		40.3	37.7	40.3	40.4	38.3	23.7	-12.72
	% impro. of Bézier from Spike2CGR		48.8	46.6	48.8	49.9	49.2	28.7	49.7
	2-Layer	FCGR	0.565	0.565	0.565	0.554	0.432	0.677	4.074
		RandmCGR	0.295	0.171	0.295	0.216	0.104	0.530	6.433
		Spike2CGR	0.461	0.454	0.461	0.433	0.278	0.604	8.932
		Bézier	0.959	0.971	0.959	0.963	0.904	0.965	13.089
	% improv. of Bézier from FCGR		39.4	40.6	39.4	40.9	47.2	28.8	-221.28
	% impro. of Bézier from Spike2CGR		49.8	51.7	49.8	53	62.6	36.1	-2922.8
	3-Layer	FCGR	0.504	0.518	0.504	0.501	0.376	0.656	4.821
		RandmCGR	0.303	0.186	0.303	0.228	0.110	0.532	8.930
		Spike2CGR	0.429	0.430	0.429	0.421	0.287	0.612	3.998
		Bézier	0.951	0.965	0.951	0.952	0.881	0.957	14.983
	% improv. of Bézier from FCGR		44.7	44.7	44.7	44.8	50.5	30.1	-210.78
	% impro. of Bézier from Spike2CGR		52.2	53.5	52.2	53.1	59.4	35.5	-274.7
	4-Layer	FCGR	0.539	0.524	0.539	0.525	0.393	0.663	5.146
		RandmCGR	0.311	0.181	0.311	0.229	0.110	0.536	10.234
		Spike2CGR	0.420	0.420	0.420	0.424	0.280	0.600	9.121
		Bézier	0.938	0.958	0.938	0.944	0.884	0.959	15.456
	% improv. of Bézier from FCGR		39.9	43.4	39.9	41.9	49.1	29.6	-200.36
	% impro. of Bézier from Spike2CGR		51.8	53.8	51.8	52	60.4	35.9	-69.4
Vision Transformer	ViT	FCGR	0.226	0.051	0.226	0.083	0.033	0.500	0.180
		RandmCGR	0.222	0.049	0.222	0.080	0.033	0.500	0.154
		Spike2CGR	0.222	0.051	0.222	0.083	0.147	0.500	0.176
		Bézier	0.462	0.254	0.462	0.327	0.147	0.572	0.160
	% improv. of Bézier from FCGR		23.6	20.3	23.6	24.4	11.4	7.2	11.11
	% impro. of Bézier from Spike2CGR		24	20.3	24	24.4	0	7.2	-9.09
Pretrained Vision Models	ResNet-50	FCGR	0.368	0.268	0.368	0.310	0.155	0.556	3.831
		RandmCGR	0.293	0.174	0.293	0.211	0.102	0.527	13.620
		Spike2CGR	0.368	0.175	0.368	0.214	0.105	0.565	10.992
		Bézier	0.964	0.967	0.964	0.961	0.907	0.948	11.415
	% improv. of Bézier from FCGR		59.6	69.9	59.6	65.1	75.2	39.2	-197.96
	% impro. of Bézier from Spike2CGR		59.6	79.2	59.6	74.7	80.2	38.3	-3.8
	VGG-19	FCGR	0.316	0.209	0.316	0.241	0.114	0.533	14.058
		RandmCGR	0.288	0.192	0.288	0.218	0.105	0.525	26.136
		Spike2CGR	0.351	0.352	0.351	0.333	0.211	0.550	19.980
		Bézier	0.896	0.879	0.896	0.873	0.680	0.840	18.837
	% improv. of Bézier from FCGR		58	67	58	63.2	56.6	30.7	-33.99
	% impro. of Bézier from Spike2CGR		54.5	52.7	54.5	56.3	46.9	29	5.7
	DenseNet	FCGR	0.081	0.006	0.081	0.012	0.013	0.500	2.001
		RandmCGR	0.094	0.008	0.094	0.016	0.015	0.500	1.974
		Spike2CGR	0.099	0.010	0.099	0.020	0.002	0.500	2.111
		Bézier	0.011	0.000	0.011	0.000	0.002	0.500	2.668
	% improv. of Bézier from FCGR		-7	-0.6	-7	-1.2	-1.1	0	-33.33
	% impro. of Bézier from Spike2CGR		-8.8	-1	-8.8	-2	0	0	-26.3
	EfficientNet	FCGR	0.100	0.088	0.100	0.094	0.035	0.532	31.194
		RandmCGR	0.284	0.107	0.284	0.152	0.078	0.500	30.223
		Spike2CGR	0.320	0.230	0.320	0.230	0.200	0.500	25.497
		Bézier	0.834	0.787	0.834	0.797	0.483	0.751	20.312
	% improv. of Bézier from FCGR		73.4	69.9	73.4	70.3	44.8	21.9	34.88
	% impro. of Bézier from Spike2CGR		51.4	55.7	51.4	56.7	28.3	25.1	20.3

Table 5.9 Classification results for different models and algorithms for **Protein Subcellular Localization dataset**. The top 5% values for each metric are underlined.

Category	DL Model	Method	Acc.	Prec.	Recall	F1 (Weig.)	F1 (Macro)	ROC AUC	Train Time (hrs.)
Tabular Models	3-Tab	OHE	0.625	0.626	0.625	0.566	0.335	0.663	0.032
		WDGRL	0.304	0.137	0.304	0.182	0.041	0.499	0.029
	4-Tab	OHE	0.613	0.478	0.613	0.534	0.323	0.662	0.067
		WDGRL	0.312	0.130	0.312	0.167	0.035	0.498	0.054
String Kernel	-	SVM	0.601	0.673	0.601	0.602	0.325	0.624	5.198
	-	NB	0.230	0.665	0.230	0.295	0.162	0.625	0.131
	-	MLP	0.647	0.696	0.647	0.641	0.302	0.628	42.322
	-	KNN	0.613	0.623	0.613	0.612	0.310	0.629	0.434
	-	RF	0.668	0.692	0.668	0.663	0.360	0.658	4.541
	-	LR	0.554	0.724	0.554	0.505	0.193	0.568	5.096
	-	DT	0.646	0.674	0.646	0.643	0.345	0.653	1.561
Custom CNN Models	1-Layer	FCGR	0.680	0.707	0.680	0.670	0.517	0.761	0.984
		Spike2CGR	0.743	0.745	0.743	0.739	0.569	0.797	0.711
		RandomCGR	0.262	0.193	0.262	0.210	0.051	0.500	8.695
		Bézier	0.652	0.652	0.652	0.644	0.592	0.766	2.698
	% improv. of Bézier from FCGR		-2.8	-5.5	-2.8	-2.6	7.5	0.5	-174.18
	2-Layer	FCGR	0.668	0.684	0.668	0.655	0.410	0.710	1.046
		Spike2CGR	0.740	0.734	0.740	0.726	0.428	0.716	0.688
		RandomCGR	0.293	0.235	0.293	0.246	0.093	0.521	8.839
		Bézier	0.656	0.669	0.656	0.644	0.610	0.778	2.976
	% improv. of Bézier from FCGR		-1.2	-1.5	-1.2	-1.1	20	6.8	-184.51
	3-Layer	FCGR	0.681	0.677	0.681	0.672	0.470	0.740	5.681
		Spike2CGR	0.729	0.729	0.729	0.715	0.354	0.677	0.831
		RandomCGR	0.320	0.102	0.320	0.155	0.028	0.500	9.440
		Bézier	0.611	0.652	0.611	0.612	0.623	0.793	4.660
	% improv. of Bézier from FCGR		-7	-2.5	-7	-6	15.3	5.3	17.97
	4-Layer	FCGR	0.624	0.617	0.624	0.606	0.262	0.623	8.991
		Spike2CGR	0.686	0.668	0.686	0.672	0.283	0.632	0.684
		RandomCGR	0.320	0.102	0.320	0.155	0.028	0.500	10.778
		Bézier	0.640	0.643	0.640	0.575	0.594	0.782	5.102
	% improv. of Bézier from FCGR		1.6	2.6	1.6	-3.1	33.2	15.9	43.25
Vision Transformer	ViT	FCGR	0.322	0.104	0.322	0.157	0.023	0.500	0.188
		Spike2CGR	0.332	0.323	0.332	0.333	0.213	0.500	0.877
		RandomCGR	0.320	0.102	0.320	0.155	0.028	0.500	0.173
		Bézier	0.316	0.100	0.316	0.152	0.022	0.500	0.183
	% improv. of Bézier from FCGR		-0.6	-0.4	-0.6	-0.5	-0.1	0	2.65
Pretrained Vision Models	ResNet-50	FCGR	0.662	0.665	0.662	0.639	0.267	0.621	8.840
		Spike2CGR	0.691	0.683	0.691	0.663	0.270	0.624	0.786
		RandomCGR	0.319	0.113	0.319	0.159	0.030	0.500	13.488
		Bézier	0.571	0.473	0.571	0.504	0.335	0.564	6.411
	% improv. of Bézier from FCGR		-9.1	-19.2	-9.1	-13.5	6.8	-5.7	27.47
	VGG-19	FCGR	0.519	0.475	0.519	0.442	0.158	0.572	3.738
		Spike2CGR	0.458	0.409	0.458	0.363	0.129	0.559	3.409
		RandomCGR	0.320	0.102	0.320	0.155	0.028	0.500	21.474
		Bézier	0.521	0.421	0.521	0.448	0.222	0.500	3.200
	% improv. of Bézier from FCGR		0.2	-5.4	0.2	0.6	6.4	-7.2	14.39
	DenseNet	FCGR	0.018	0.000	0.018	0.001	0.018	0.500	2.566
		Spike2CGR	0.017	0.000	0.017	0.000	0.001	0.500	2.675
		RandomCGR	0.015	0.000	0.015	0.000	0.001	0.500	2.123
		Bézier	0.011	0.000	0.011	0.001	0.011	0.500	2.332
	% improv. of Bézier from FCGR		-0.8	0	-0.8	0	-0.8	0	9.11
	EfficientNet	FCGR	0.169	0.028	0.169	0.049	0.013	0.500	34.443
		Spike2CGR	0.169	0.031	0.169	0.053	0.015	0.500	31.229
		RandomCGR	0.317	0.108	0.317	0.162	0.032	0.529	37.334
		Bézier	0.465	0.427	0.465	0.394	0.157	0.577	35.768
	% improv. of Bézier from FCGR		29.6	39.9	29.6	34.5	14.4	7.7	-3.84

Table 5.10 Classification results for different models and algorithms for **Coronavirus Host dataset**. The top 5% values for each metric are underlined.

Category	DL Model	Method	Acc.	Prec.	Recall	F1 (Weig.)	F1 (Macro)	ROC AUC	Train Time (hrs.)
Tabular Models	3-Tab	OHE	0.768	0.839	0.768	0.790	0.452	0.719	0.042
		WDGRL	0.615	0.740	0.615	0.660	0.326	0.603	0.0001
	4-Tab	OHE	0.796	0.843	0.796	0.807	0.474	0.736	0.056
		WDGRL	0.631	0.754	0.631	0.673	0.346	0.623	0.0002
String Kernel	-	SVM	0.802	0.836	0.802	0.813	0.454	0.692	0.789
	-	NB	0.872	0.869	0.872	0.864	0.523	0.732	0.018
	-	MLP	0.611	0.771	0.611	0.666	0.348	0.626	2.478
	-	KNN	0.871	0.849	0.871	0.853	0.482	0.694	0.286
	-	RF	0.866	0.837	0.866	0.847	0.470	0.681	1.029
	-	LR	0.881	0.872	0.881	0.870	0.536	0.720	0.254
	-	DT	0.835	0.843	0.835	0.838	0.465	0.702	0.338
Custom CNN Models	1-Layer	FCGR	0.863	0.831	0.863	0.844	0.490	0.677	0.357
		Spike2CGR	0.783	0.613	0.783	0.687	0.219	0.500	0.999
		RandomCGR	0.792	0.638	0.792	0.707	0.221	0.497	0.404
		Bézier	0.835	0.779	0.835	0.781	0.314	0.548	0.805
	% improv. of Bézier from FCGR		-2.8	-5.2	-2.8	-6.3	-17.6	-12.9	-125.49
	2-Layer	FCGR	0.852	0.833	0.852	0.837	0.489	0.676	0.419
		Spike2CGR	0.783	0.613	0.783	0.687	0.219	0.500	1.196
		RandomCGR	0.800	0.640	0.800	0.711	0.222	0.500	0.389
		Bézier	0.814	0.795	0.814	0.803	0.419	0.633	0.626
	% improv. of Bézier from FCGR		-3.8	-3.8	-3.8	-3.4	-7	-4.3	-49.40
	3-Layer	FCGR	0.800	0.640	0.800	0.711	0.222	0.500	0.490
		Spike2CGR	0.783	0.612	0.783	0.687	0.219	0.500	1.456
		RandomCGR	0.800	0.640	0.800	0.711	0.222	0.500	0.391
		Bézier	0.830	0.748	0.830	0.780	0.296	0.541	0.637
	% improv. of Bézier from FCGR		3	10.8	3	6.9	7.4	4.1	-30
	4-Layer	FCGR	0.831	0.735	0.831	0.779	0.329	0.586	0.498
		Spike2CGR	0.783	0.612	0.783	0.687	0.219	0.500	1.776
		RandomCGR	0.800	0.640	0.800	0.711	0.222	0.500	0.435
		Bézier	0.825	0.681	0.825	0.746	0.226	0.500	0.668
	% improv. of Bézier from FCGR		-0.6	-5.4	-0.6	-3.3	-10.3	-8.6	-34.13
Vision Transformer	ViT	FCGR	0.767	0.588	0.767	0.666	0.217	0.500	0.031
		Spike2CGR	0.754	0.487	0.74	0.565	0.211	0.500	0.650
		RandomCGR	0.756	0.512	0.756	0.632	0.201	0.500	0.032
		Bézier	0.825	0.681	0.825	0.746	0.226	0.500	0.027
	% improv. of Bézier from FCGR		5.8	9.3	5.8	8	0.9	0	12.90
Pretrained Vision Models	ResNet-50	FCGR	0.800	0.642	0.800	0.712	0.222	0.501	1.317
		Spike2CGR	0.770	0.559	0.770	0.654	0.198	0.500	2.290
		RandomCGR	0.800	0.640	0.800	0.711	0.222	0.500	1.387
		Bézier	0.835	0.780	0.835	0.796	0.334	0.601	0.175
	% improv. of Bézier from FCGR		3.5	13.8	3.5	8.4	11.2	10	86.71
	VGG-19	FCGR	0.803	0.684	0.803	0.720	0.243	0.509	1.189
		Spike2CGR	0.765	0.650	0.765	0.650	0.200	0.500	2.111
		RandomCGR	0.800	0.640	0.800	0.711	0.222	0.500	1.054
		Bézier	0.825	0.681	0.825	0.746	0.226	0.500	2.144
	% improv. of Bézier from FCGR		2.2	-0.3	2.2	2.6	-1.7	-0.9	-80.31
	DenseNet	FCGR	0.116	0.013	0.116	0.024	0.052	0.500	0.987
		Spike2CGR	0.116	0.011	0.116	0.022	0.050	0.500	1.767
		RandomCGR	0.095	0.011	0.095	0.010	0.095	0.500	1.381
		Bézier	0.105	0.011	0.105	0.020	0.105	0.500	1.211
	% improv. of Bézier from FCGR		-1.1	-0.2	-1.1	-0.4	5.3	0	-22.69
	EfficientNet	FCGR	0.089	0.008	0.089	0.014	0.041	0.500	1.622
		Spike2CGR	0.085	0.005	0.085	0.009	0.008	0.500	2.221
		RandomCGR	0.028	0.002	0.028	0.004	0.027	0.500	1.988
		Bézier	0.058	0.003	0.058	0.006	0.027	0.500	1.566
	% improv. of Bézier from FCGR		-3.1	-0.5	-3.1	-0.8	-1.4	0	3.45

Table 5.11 Classification results for different models and algorithms for **ACPs (Breast Cancer) dataset**. The top 5% values for each metric are underlined.

Category	DL Model	Method	Acc.	Prec.	Recall	F1 (Weig.)	F1 (Macro)	ROC AUC	Train Time (hrs.)
Tabular Models	3-Tab	OHE	0.627	0.699	0.627	0.613	0.566	0.729	0.024
		WDGRL	0.657	0.716	0.657	0.649	0.601	0.758	0.020
	4-Tab	OHE	0.680	0.704	0.680	0.661	0.581	0.762	0.042
		WDGRL	0.654	0.692	0.654	0.635	0.551	0.743	0.038
String Kernel	-	SVM	0.618	0.617	0.618	0.613	0.588	0.753	39.791
	-	NB	0.338	0.452	0.338	0.347	0.333	0.617	0.276
	-	MLP	0.597	0.595	0.597	0.593	0.549	0.737	331.068
	-	KNN	0.645	0.657	0.645	0.646	0.612	0.774	1.274
	-	RF	0.731	0.776	0.731	0.729	0.723	0.808	12.673
	-	LR	0.571	0.570	0.571	0.558	0.532	0.716	2.995
	-	DT	0.630	0.631	0.630	0.630	0.598	0.767	2.682
Custom CNN Models	1-Layer	FCGR	0.717	0.719	0.717	0.709	0.711	0.834	0.351
		RandmCGR	0.820	0.827	0.820	0.816	0.787	0.872	0.355
		Spike2CGR	0.662	0.698	0.662	0.660	0.627	0.768	0.353
		Bézier	0.710	0.712	0.710	0.700	0.713	0.831	0.339
	% impro. of Bézier from Spike2CGR		5.1	1.4	5.1	4	8.6	6.3	-3.9
	2-Layer	FCGR	0.705	0.708	0.705	0.694	0.691	0.831	0.365
		RandmCGR	0.785	0.791	0.785	0.782	0.750	0.845	0.622
		Spike2CGR	0.665	0.685	0.665	0.664	0.633	0.786	0.692
		Bézier	0.700	0.722	0.700	0.695	0.659	0.803	0.350
	% impro. of Bézier from Spike2CGR		3.5	3.7	3.5	3.1	2.6	1.7	49.4
	3-Layer	FCGR	0.632	0.641	0.632	0.623	0.609	0.767	0.332
		RandmCGR	0.710	0.724	0.710	0.697	0.661	0.807	0.530
		Spike2CGR	0.580	0.636	0.580	0.582	0.514	0.715	0.331
		Bézier	0.426	0.498	0.426	0.351	0.298	0.594	0.376
	% impro. of Bézier from Spike2CGR		-15.4	-13.8	-15.4	-23.1	-21.6	-12.1	-13.59
	4-Layer	FCGR	0.300	0.090	0.300	0.138	0.065	0.500	0.331
		RandmCGR	0.287	0.082	0.287	0.128	0.063	0.500	0.521
		Spike2CGR	0.377	0.385	0.377	0.305	0.232	0.562	0.311
		Bézier	0.313	0.097	0.313	0.149	0.068	0.500	0.321
	% impro. of Bézier from Spike2CGR		-6.4	-28.8	-6.4	-15.6	-16.4	-6.2	-3.2
Vision Transformer	ViT	FCGR	0.300	0.090	0.300	0.138	0.065	0.500	0.782
		RandmCGR	0.295	0.140	0.295	0.142	0.097	0.510	0.828
		Spike2CGR	0.307	0.094	0.307	0.144	0.067	0.500	3.787
		Bézier	0.382	0.326	0.382	0.323	0.239	0.613	0.654
	% impro. of Bézier from Spike2CGR		7.5	23.2	7.5	17.9	17.2	11.3	82.7
Pretrained Vision Models	ResNet-50	FCGR	0.357	0.251	0.357	0.283	0.208	0.500	0.495
		RandmCGR	0.290	0.192	0.290	0.137	0.072	0.500	0.481
		Spike2CGR	0.352	0.341	0.352	0.295	0.208	0.565	2.443
		Bézier	0.408	0.244	0.408	0.294	0.184	0.561	0.873
	% impro. of Bézier from Spike2CGR		5.6	-9.7	5.6	-0.1	-2.4	-0.4	64.2
	VGG-19	FCGR	0.345	0.285	0.345	0.249	0.181	0.540	1.078
		RandmCGR	0.287	0.082	0.287	0.128	0.063	0.500	1.115
		Spike2CGR	0.307	0.094	0.307	0.144	0.067	0.500	3.032
		Bézier	0.317	0.132	0.317	0.176	0.098	0.510	1.221
	% impro. of Bézier from Spike2CGR		1	3.8	1	3.2	3.1	1	59.7
	DenseNet	FCGR	0.075	0.005	0.075	0.010	0.019	0.500	0.764
		RandmCGR	0.062	0.003	0.062	0.007	0.016	0.500	0.825
		Spike2CGR	0.067	0.004	0.067	0.008	0.018	0.500	1.295
		Bézier	0.078	0.007	0.078	0.013	0.022	0.491	0.822
	% impro. of Bézier from Spike2CGR		1.1	0.3	1.1	0.5	0.4	-0.9	36.5
	EfficientNet	FCGR	0.200	0.141	0.200	0.147	0.094	0.517	0.814
		RandmCGR	0.287	0.082	0.287	0.128	0.063	0.500	0.837
		Spike2CGR	0.275	0.169	0.275	0.187	0.112	0.524	1.343
		Bézier	0.313	0.097	0.313	0.149	0.068	0.500	0.844
	% impro. of Bézier from Spike2CGR		3.8	-7.2	3.8	-3.8	-4.4	-2.4	37.1

Table 5.12 Classification results for different models and algorithms for **Human DNA dataset**. The top 5% values for each metric are underlined.

Category	DL Model	Method	Acc.	Prec.	Recall	F1 (Weig.)	F1 (Macro)	ROC AUC	Train Time (hrs.)
Tabular Models	3-Tab	OHE	0.966	0.935	0.966	0.950	0.098	0.500	0.132
		WDGRL	0.966	0.935	0.966	0.950	0.098	0.500	0.001
	4-Tab	OHE	0.966	0.935	0.966	0.950	0.098	0.500	0.155
		WDGRL	0.966	0.935	0.966	0.950	0.098	0.500	0.001
String Kernel	-	SVM	0.812	0.813	0.812	0.811	0.084	0.502	10.254
	-	NB	0.537	0.643	0.537	0.549	0.096	0.502	1.24
	-	MLP	0.789	0.788	0.789	0.790	0.079	0.505	13.149
	-	KNN	0.844	0.858	0.844	0.842	0.087	0.503	2.348
	-	RF	0.929	0.927	0.929	0.925	0.098	0.507	9.315
	-	LR	0.772	0.769	0.772	0.760	0.073	0.502	5.652
	-	DT	0.834	0.829	0.834	0.832	0.075	0.508	3.318
Custom CNN Models	1-Layer	RandmCGR	0.962	0.926	0.962	0.944	0.098	0.500	0.988
		Bézier	0.970	0.942	0.970	0.956	0.109	0.512	1.003
	% impro. of Bézier from RandomCGR		0.8	1.6	0.8	1.2	1.1	12	-1.51
	2-Layer	RandmCGR	0.962	0.926	0.962	0.944	0.098	0.500	0.989
		Bézier	0.970	0.942	0.970	0.956	0.109	0.512	1.253
	% impro. of Bézier from RandomCGR		0.8	1.6	0.8	1.2	1.1	12	-26.6
	3-Layer	RandmCGR	0.962	0.926	0.962	0.944	0.098	0.500	1.411
		Bézier	0.970	0.942	0.970	0.956	0.109	0.511	1.082
	% impro. of Bézier from RandomCGR		0.8	1.6	0.8	1.2	1.1	12	80.04
	4-Layer	RandmCGR	0.962	0.926	0.962	0.944	0.098	0.500	1.331
		Bézier	0.970	0.942	0.970	0.956	0.109	0.512	1.210
	% impro. of Bézier from RandomCGR		0.8	1.6	0.8	1.2	1.1	12	9.09
Vision Transformer	ViT	RandmCGR	0.962	0.926	0.962	0.944	0.098	0.500	1.876
		Bézier	0.970	0.942	0.970	0.956	0.109	0.512	1.864
	% impro. of Bézier from RandomCGR		0.8	1.6	0.8	1.2	1.1	12	0.63
Pretrained Vision Models	ResNet-50	RandmCGR	0.962	0.926	0.962	0.944	0.098	0.500	1.872
		Bézier	0.970	0.940	0.970	0.950	0.100	0.500	1.142
	% impro. of Bézier from RandomCGR		0.8	1.4	0.8	0.6	0.2	0	38.99
	VGG-19	RandmCGR	0.962	0.926	0.962	0.944	0.098	0.500	7.120
		Bézier	0.970	0.940	0.970	0.950	0.100	0.500	2.899
	% impro. of Bézier from RandomCGR		0.8	1.4	0.8	0.6	0.2	0	59.2
	DenseNet	RandmCGR	0.001	0.024	0.001	0.004	0.000	0.500	5.043
		Bézier	0.001	0.023	0.001	0.066	0.000	0.500	2.867
	% impro. of Bézier from RandomCGR		0	1	0	6.2	0	0	43.14
	EfficientNet	RandmCGR	0.962	0.926	0.962	0.944	0.098	0.500	4.892
		Bézier	0.969	0.938	0.969	0.950	0.100	0.500	3.892
	% impro. of Bézier from RandomCGR		0.6	1.2	0.6	5.6	0.2	0	20.44

Table 5.13 Classification results for different models and algorithms for **SMILES String dataset**. The best value for each metric is underlined. As the performances of most of the models are the same and highlighting the top 5% includes a lot of data, that's why we only underlined the best one.

Category	DL Model	Method	Acc.	Prec.	Recall	F1 (Weig.)	F1 (Macro)	ROC AUC	Train Time (hrs.)
Tabular Models	3-Tab	OHE	0.515	0.682	0.515	0.611	0.451	0.652	0.003
		WDGRL	0.599	0.600	0.599	0.565	0.515	0.659	0.001
	4-Tab	OHE	0.516	0.555	0.516	0.511	0.451	0.652	0.003
		WDGRL	0.612	0.588	0.612	0.588	0.530	0.670	0.001
String Kernel	-	SVM	0.301	0.322	0.301	0.294	0.294	0.615	0.886
	-	NB	0.369	0.376	0.369	0.357	0.352	0.649	0.039
	-	MLP	0.219	0.231	0.219	0.212	0.211	0.568	3.476
	-	KNN	0.400	0.409	0.400	0.388	0.387	0.669	0.169
	-	RF	0.341	0.354	0.341	0.334	0.333	0.638	1.478
	-	LR	0.397	0.397	0.397	0.389	0.386	0.666	21.209
	-	DT	0.283	0.290	0.283	0.282	0.281	0.603	0.392
Custom CNN Models	1-Layer	RandmCGR	0.989	0.989	0.989	0.989	0.989	0.989	0.400
		Bézier	0.957	0.953	0.957	0.953	0.844	0.919	0.312
	% improv. from RandomCGR	of Bézier	-3.2	-3.6	-3.2	-3.6	-14.5	-7	22
	2-Layer	RandmCGR	0.985	0.985	0.985	0.985	0.985	0.992	0.4121
		Bézier	0.943	0.941	0.943	0.939	0.827	0.911	0.345
	% improv. from RandomCGR	of Bézier	-4.2	-4.3	-4.2	-4.6	-15.8	-1.1	16.2
	3-Layer	RandmCGR	0.085	0.007	0.085	0.013	0.015	0.500	0.541
		Bézier	0.886	0.893	0.886	0.882	0.789	0.887	0.453
	% improv. from RandomCGR	of Bézier	80.1	88.6	80.1	86.9	77.4	38.7	16.2
	4-Layer	RandmCGR	0.155	0.044	0.155	0.063	0.074	0.545	0.554
		Bézier	0.900	0.908	0.900	0.897	0.802	0.895	0.438
	% improv. from RandomCGR	of Bézier	74.5	86.4	74.5	83.4	72.8	35	20.9
Vision Transformer	ViT	RandmCGR	0.110	0.012	0.110	0.021	0.019	0.500	0.807
		Bézier	0.099	0.009	0.099	0.017	0.022	0.500	1.090
	% improv. from RandomCGR	of Bézier	-1.1	-0.3	-1.1	-0.4	-0.3	0	-23.9
Pretrained Vision Models	ResNet-50	RandmCGR	0.525	0.608	0.525	0.485	0.496	0.740	0.653
		Bézier	0.546	0.545	0.546	0.479	0.457	0.728	0.543
	% improv. from RandomCGR	of Bézier	2.1	-6.3	2.1	0.6	-3.9	-1.2	16.8
	VGG-19	RandmCGR	0.410	0.421	0.410	0.334	0.410	0.673	1.220
		Bézier	0.843	0.867	0.843	0.838	0.741	0.856	1.421
	% improv. from RandomCGR	of Bézier	43.3	44.6	43.3	50.4	33.1	18.3	-16.47
	DenseNet	RandmCGR	0.080	0.056	0.080	0.052	0.053	0.489	2.118
		Bézier	0.113	0.130	0.113	0.043	0.049	0.508	2.332
	% improv. from RandomCGR	of Bézier	3.3	7.4	3.3	-0.9	-0.4	1.9	-10.10
	EfficientNet	RandmCGR	0.735	0.719	0.735	0.697	0.689	0.851	1.011
		Bézier	0.929	0.928	0.929	0.924	0.808	0.898	0.889
	% improv. from RandomCGR	of Bézier	19.4	20.9	19.4	22.7	11.9	4.7	12.06

Table 5.14 Classification results for different models and algorithms for **Music Genre dataset**. The top 5% values for each metric are underlined.

Dataset	Method	AUC
Coronavirus Host	OHE	0.3914
	Spike2Vec	0.4054
	PWM2Vec	0.4169
	PSSMFreq2Vec	0.4029
	PSSM2Vec	**0.4417**
SARS-CoV-2	OHE	0.2248
	Spike2Vec	0.2549
	PWM2Vec	**0.2850**
	PSSMFreq2Vec	0.2554
	PSSM2Vec	0.2819

Table 5.16 k-ary neighborhood agreement for $k = 1$ to 99.

Dataset	No. of Seq.	Method	Runtime
Coronavirus Host	5558	OHE	196.31 Sec.
		Spike2Vec	1179.66 Sec.
		PWM2Vec	1506.63 Sec.
		Approx. Kernel	379.47 Sec.
		PSSMFreq2Vec	908.12 Sec.
		PSSM2Vec	48.25 Sec.
SARS-CoV-2	2519386	OHE	$>$ 3 days
		Spike2Vec	$>$ 3 days
		PWM2Vec	$>$ 3 days
		PSSMFreq2Vec	$>$ 3 days
		PSSM2Vec	$>$ 4 Hours

Table 5.17 Runtime for generating feature vectors using different methods.

CHAPTER 6

Conclusion

This work deals with proposing methods to transform bio-sequence data into ML/DL compatible form. In this regard, two categories of alignment-free methods are put forward, feature-engineering-based and image-based encoding methods. Among the feature-engineering-based methods, PSSM2Vec and PSSMFre2Vec are scalable and compact, and Hashing2Vec is fast in terms of embedding generation runtime. Moreover, the image-based encoding methods have enabled the application of DL models on bio-sequence classification.

REFERENCES

Ali, S., Bello, B., Chourasia, P., Punathil, R. T., Zhou, Y., & Patterson, M. 2022, MDPI Biology

Ali, S., & Patterson, M. 2021, in International Conference on Big Data (Big Data), 1533–1540

Ali, S., Sahoo, B., Ullah, N., Zelikovskiy, A., Patterson, M., & Khan, I. 2021, in International Symposium on Bioinformatics Research and Applications, 153–164

Barnsley, M. F. 2012, Fractals Everywhere: New Edition

Baydas, S., & Karakas, B. 2019, Journal of Taibah University for Science, 13, 522

Brandes, N., Ofer, D., Peleg, Y., Rappoport, N., & Linial, M. 2022, Bioinformatics, 38, 2102

Buchan, D. W., & Jones, D. T. 2019, Nucleic acids research, 47, W402

Das, K. 2012, Journal of medicinal chemistry, 55, 6263

Devijver, P., & Kittler, J. 1982, in London, GB: Prentice-Hall, 1–448

Dhar, S., Zhang, C., Mandoiu, I., & Bansal, M. S. 2020, in International Symposium on Bioinformatics Research and Applications, 203–216

Dong, G., & Pei, J. 2007, Sequence data mining, Vol. 33 (Springer Science & Business Media)

Farhan, M., Tariq, J., Zaman, A., Shabbir, M., & Khan, I. 2017, in Advances in neural information processing systems (NeurIPS) (.), 6935–6945

Fowler, G., Noll, L. C., Vo, K.-P., Eastlake, D., & Hansen, T. 2011, IETF-draft: Fremont, CA, USA

Ghandi, M., Lee, D., Mohammad-Noori, M., & Beer, M. A. 2014, PLoS computational

biology, 10, e1003711

Grisoni, et al. '2019', 'Journal of Molecular Modeling', '25', '112'

Hadfield, J. et al. 2018, Bioinformatics, 34, 4121

Han, X.-A., Ma, Y., & Huang, X. 2008, Journal of Computational and Applied Mathematics, 217, 180

He, K., Zhang, X., Ren, S., & Sun, J. 2016, in IEEE conference on computer vision and pattern recognition, 770–778

Heinzinger, M., Elnaggar, A., Wang, Y., Dallago, C., Nechaev, D., Matthes, F., & Rost, B. 2019, BMC bioinformatics, 20, 1

Hoang, T., Yin, C., & Yau, S. S.-T. 2016, Genomics, 108, 134

Hug, R., Hübner, W., & Arens, M. 2020in , 10162–10169

Human DNA. 2022, `https://www.kaggle.com/code/nageshsingh/demystify-dna-sequencing-with-machine-learning/data`, [Online; accessed 10-October-2022]

Iandola, F., Moskewicz, M., Karayev, S., Girshick, R., Darrell, T., & Keutzer, K. 2014, arXiv preprint arXiv:1404.1869

Jeffrey, H. J. 1990, Nucleic acids research, 18, 2163

Kuzmin, K., et al. 2020, Biochemical and Biophysical Research Communications, 533, 553

Li, T., Ogihara, M., & Li, Q. 2003, in Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval, 282–289

Liu, Y., Shen, C., Jin, L., He, T., Chen, P., Liu, C., & Chen, H. 2021, IEEE Transactions

on Pattern Analysis and Machine Intelligence, 44, 8048

Löchel, H. F., Eger, D., Sperlea, T., & Heider, D. 2020, Bioinformatics, 36, 272

Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., & Watkins, C. 2002, Journal of machine learning research, 2, 419

Ma, Y., Yu, Z., Tang, R., Xie, X., Han, G., & Anh, V. V. 2020, Entropy, 22, 255

Minh, B. Q., et al. 2020, Molecular Biology and Evolution, 37, 1530

Morales, A., et al. 2021, Genome biology and evolution, 13

Nishida, K., Frith, M. C., & Nakai, K. 2009, Nucleic acids research, 37, 939

Pedersen, S. F., Ho, Y.-C., et al. 2020, The Journal of clinical investigation, 130, 2202

Pickett, B. E. et al. 2012, Nucleic acids research, 40, D593

Qiao, L., Ding, W., Qiu, X., & Zhang, C. 2023, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 13218–13228

Rao, V. S., Srinivas, K., Sujini, G., & Kumar, G. 2014, International journal of proteomics, 2014

Rizzo, R., Fiannaca, A., La Rosa, M., & Urso, A. 2016, in Proceedings of the 17th International Conference on Computer Systems and Technologies 2016, 222–228

Roberts, M., Haynes, W., Hunt, B., Mount, S., & Yorke, J. 2004, Bioinformatics, 20, 3363

Rognan, D. 2007, British journal of pharmacology, 152, 38

Shamay, Y. et al. 2018, Nature materials, 17, 361

Shen, J., Qu, Y., Zhang, W., & Yu, Y. 2018, in AAAI conference on artificial intelligence

Shrikumar, A., Prakash, E., & Kundaje, A. 2019, Bioinformatics, 35, i173

Simonyan, K., & Zisserman, A. 2015, in International Conference on Learning Representations

Singh, R., Sekhon, A., et al. 2017, in Joint ECML and Knowledge Discovery in Databases, 356–373

Stormo, G. D., Schneider, T. D., Gold, L., & Ehrenfeucht, A. 1982, Nucleic Acids Research, 10, 2997

Strichartz, R. S. 2000, The American Mathematical Monthly, 107, 316

Strodthoff, N., Wagner, P., Wenzel, M., & Samek, W. 2020, Bioinformatics, 36, 2401

Tan, M., & Le, Q. 2019, in International conference on machine learning, PMLR, 6105–6114

Tayebi, Z., Ali, S., & Patterson, M. 2021, Algorithms, 14, 348

Tzanov, V. 2015, arXiv preprint arXiv:1502.01384

Uyangodage, L., Ranasinghe, T., & Hettiarachchi, H. 2021, in NLP for Internet Freedom: Censorship, Disinformation, and Propaganda, 130–135

Van der Maaten, L., & Hinton, G. 2008, Journal of machine learning research, 9

Wold, S., Esbensen, K., & Geladi, P. 1987, Chemometrics and intelligent laboratory systems, 2, 37

Xie, J., Girshick, R., & Farhadi, A. 2016, in International conference on machine learning, 478–487

Yao, et al. 2019, in Proceedings of the Future Technologies Conference, Springer, 276–282

Zhu, Y., & Ting, K. M. 2021, Journal of Artificial Intelligence Research, 71, 667