Deep Learning Methods for Planted Motif Discovery in Genomic Data

Wajih Abdul Ghani Abdul Hussain[1], Hussein Keitan Al-Khafaji[2] and Thekra Abbas[1]
[1]*Department of Computer Science, College of Science, Mustansiriyah University, 10052 Baghdad, Iraq*
[2]*Department of Computer Communication Engineering, Uruk University, 10071 Baghdad, Iraq*
wajeeh.abdulgani@uomustansiriyah.edu.iq, hussein.k.khafaji@uruk.edu.iq, thekra.abbas@uomustansiriyah.edu.iq.

Keywords: Bioinformatics, Planted Motif Problem, Approximate Algorithms, Deep Learning, Convolutional Neural Network.

Abstract: The massive growth of genomic data has made it necessary to have effective computational algorithms that are able to detect small, biologically significant sequence patterns which are called motifs. The Planted Motif Problem (PMP) that consists of locating (l, d) motif in a number of sequences with a maximum of d mutations is still computationally challenging because of its combinatorial complexity. PMM (Planted Motif Miner) is a hybrid approximate framework that we suggest to identify (l, d) planted motifs efficiently using pattern mining and deep learning. In this work, PMM has seven steps including data preprocessing, k-mer segmentation, frequent-pattern aggregation, motif extension, consensus motif mining, d-neighbor generation, and CNN-based classification module. The model uses multiprocessing and random sampling to deal with the exponential growth of d-neighbors with large (l, d). Experimental results on simulated and real data show that PMM are accurate ranging from (87% to 99%) based on length of (l, d) parameters and better than state of the art motif discovery algorithms, such as qPMS10, FMotif, GADEM, and MEME-ChIP. Its performance advantage is obvious on large instances where the current ways will normally fail or take over 24 hours to accomplish. In addition, the suggested algorithm can execute large (l, d) like (30, 10). The findings indicate that the suggested method is able to obtain > 99% accuracy and F1-scores above 97% with a minimum loss rate, when using moderate parameter configurations, such as (18, 6) and (26, 6). The sampling is necessary when the problem complexity is higher to make sure that it is feasible, the classification performance tends to decrease, but the CNN can still achieve strong accuracy and F1-scores over 82% in the most difficult settings.

1 INTRODUCTION

The human genome contains nearly 3 billion nucleotides, making manual analysis and motif detection impractical and costly in clinical contexts [1]. Since laboratory experiments are both expensive and time-consuming, computational data mining techniques are needed to efficiently identify functional motifs with high accuracy. Developing reliable methods for motif discovery is therefore a key challenge in bioinformatics. [2], [3].

Motif discovery is especially important for locating transcription factor binding sites (TFBS) in DNA sequences. These binding sites are generally short and can vary slightly between occurrences, which makes detecting them difficult [4]. A commonly studied formulation of this paper is the Planted Motif Problem (PMP), also known as the (l, d)-motif problem. In this problem, given several DNA sequences and two integers l (motif length) and d (allowed mismatches), the goal is to find an l-length pattern that appears in most sequences with at most d differences.

For example, in the sequences CATACGT, ACAAGTC, and AATCGTG, the substring CAT is considered a valid motif when l = 3 and d = 1, since it appears in each sequence with no more than one mismatch.

While exact algorithms can guarantee complete motif detection, they are often computationally expensive. Approximate algorithms are faster but the problem remains NP-hard. Therefore, more efficient approaches are needed.

This work addresses the Planted Motif Problem using Convolutional Neural Networks (CNNs). CNNs are advantageous for motif discovery because they can automatically learn meaningful patterns from data, scale to large genomic datasets, and

perform well even when the sequence data is noisy or complex.

2 RELATED WORKS

This paragraph underscores how motif discovery algorithms have changed throughout the years, initially using exact motif discovery algorithms such as PMS1, then providing approximation motif discovery algorithms such as qPMS10, and finally providing deep learning-based motif discovery algorithms such as iDeeps. The ever-improving space efficiency, processing speed and scalability highlights the dynamicity of the present research field.

In [5], the authors suggested algorithms PMSi and PMSP which were based on the ideas used in PMS1. These algorithms provided a better space efficiency than PMS1 and thus permitted the improved running times in diverse situations. It is worth to note that PMSP is better than PMS1 and PMSi in challenging cases, as it can perform better. It is also a better performer than existing exact algorithms because it has a much smaller memory requirement, and can solve previously intractable instances [6].

PMS1 is effective in solving problems like (9, 2), (11, 3) and (13, 4) within a few minutes and voter method is effective in solving the problem as well as (15, 5) in 22 minutes. The algorithms, however, have increased memory requirements with increased value of d. It is noteworthy that PMSP proves to be effective in dealing with demanding situations like (l, d) = (15, 5) one. Motif discovery algorithms have over time shifted to less precise methods such as PMS1, PMSprune, stemming, and variants PMS4 through PMS8, to more recent approximate methods such as qPMS9 and qPMS10, capable of working on complex (l, d) parameters such as (30, 15) [6]. This transition to deep learning as the alternative to traditional methods of bioinformatics is the essential change in the sphere. Neural networks enable the scientists to handle the huge volumes and complexity of modern biological data with additional accuracy and speed. With the further development of these deep learning models, they will become the key to finding the biological motifs in an unprecedented precision.

The famous algorithm that dealing with deep learning are as follows:

DanQ [7] uses a CNN layer followed by a bidirectional LSTM (BLSTM). The CNN layer first detects motif locations in the sequence, and after max-pooling, the BLSTM layer captures long-range dependencies.

KEGRU [8] replaces the CNN with k-mer features combined with a single GRU layer. By relying on k-mer representations instead of convolution, KEGRU effectively models sequence relationships and performs well in RNA motif identification.

iDeeps [9] integrates both CNN and BLSTM networks to learn RNA sequence and structural motifs simultaneously. The CNN identifies interpretable binding motifs, while the BLSTM improves recognition of both sequence patterns and structural features.

Compared to these works, our model will address the limitations of high memory usage and scalability of earlier exact algorithms by introducing a novel model and pruning strategy. Our proposed PMM model achieves accept space efficiency, competitive running time, accuracy in results. According to the survey, deep learning techniques have not been applied to the planted motif mining problem, despite their inherent convolutional strengths. Furthermore, the absence of recent studies in this area emphasizes the originality and contribution of the proposed work.

3 BIOLOGICAL BACKGROUND

The genome is the complete hereditary information of an organism, stored in its DNA [10]. DNA is a double-stranded molecule made of four bases-A, T, C, and G-which pair as A-T and C-G. In eukaryotic cells, DNA is organized into chromosomes located in the nucleus. Its main function is to store and transmit genetic information. This occurs through two key processes: replication, where DNA makes identical copies of itself, and gene expression, where genetic information is used to produce proteins through transcription and translation as shown in Figure 1.

4 MACHINE LEARNING AND DEEP LEARNING IN GENOMICS

The age of big data has necessitated the derivation of useful knowledge out of big data and also in bioinformatics [11]. This has been extensively done using traditional machine learning purpose, although it relies a great deal on manually drawn features, and these take professional expertise are sometimes hard and laborious to choose [12]. In addition, the classical machine learning is incapable of processing raw data and has trouble processing data of high dimensions

like DNA sequences [13]. Large data, powerful computing and improved algorithms have made possible deep learning frees itself of such limitations by learning useful representations of raw data, automatically [14], [15]. Existing deep learning methods in motif mining can be roughly divided into three types of models: convolutional neural network (CNN) based models, recurrent neural network (RNN) based models, and hybrid CNN–RNN based models [16].

4.1 CNN Framework

Convolutional Neural Networks (CNNs) consist of interconnected layers of neurons, typically including convolutional, pooling, and fully connected layers [17]. CNNs learn simple features in early layers and increasingly complex patterns in deeper layers. Two key properties make them efficient: weight sharing, where the same weights are used across a layer, and local connectivity [18], where neurons process only small regions of the input.

1) Convolutional Layers. These layers apply kernels (e.g., 3×3 or 5×5) across the input to produce feature maps [19]. Each kernel detects a specific local pattern, and activation functions (such as ReLU or tanh) introduce nonlinearity [20].
2) Pooling Layers. These layers reduce the spatial dimensions of feature maps, lowering computation. Max-pooling selects the highest value in a region, while average pooling computes the mean [21].
3) Fully Connected Layers. These layers combine extracted features and map them to output classes, often using softmax for classification [22].

5 PROPOSED PLANTED MOTIF MINER (PMM)

The proposed architecture explained in this paper has seven main stages including data pre-processing, k-mer segmentation, popular k-mer mining, motif joining and extension, consensus motif mining, generation of d-neighbors and CNN model for motif classification.

The methods presented in the architecture is not an exact method but the probability of finding the correct or near to correct planted motif is quite high. The prediction of the model is formulated as a binary classification task whose output represents whether a planted motif in a DNA sequence of length L is present (1) or not (0). The steps of architecture are illustrated in Figure 2.

5.1 Data Preprocessing

At this stage, the data will go through several stages:

1) Undesirable data Elimination. The dataset was checked for mistakes that might have occurred in the dataset. These mistakes include special characters, numbers, and lowercase letters. All these errors are removed from the dataset.
2) Type Checking. A type checker process was used to determine whether the data set was made of DNA (A, C, G, and T), RNA (A, C, G, and U), or protein (A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, and Y), bases.

Binary encoding. Binary encoding is used to transform categorical data such as nucleotides into a numerical form. Each nucleotide in DNA can be represented as a two bit encoding, for example, A is encoded as (0,0), C as (1,0), G as (0,1), and T as (1,1).

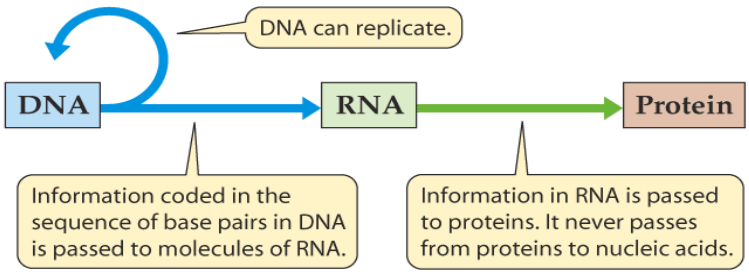

Figure 1: Transcription and translation.

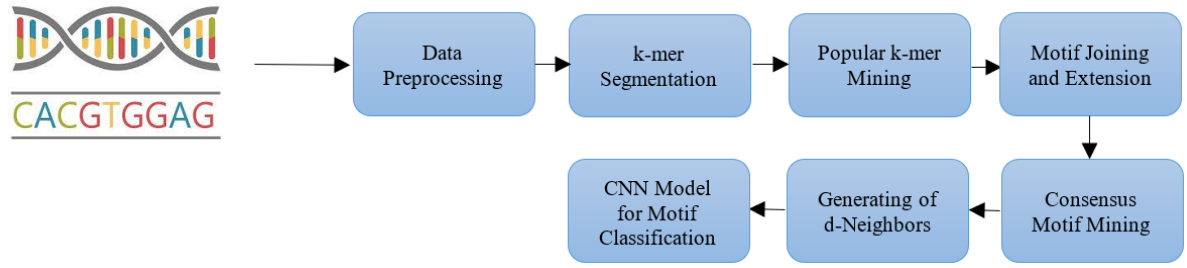

Figure 2: Planted motif model using deep learning.

5.2 k-mer Segmentation

In this step, all sequences in the data set are segmented with k-length subsequences (called k-mers) where each S_i belong to the dataset where ($0 < i < t$) and t represents the count of sequences in the data set, and the count of k-mers will have $|Si| - l + 1$ bases. As a first step all the sub-sequences corresponding to window size k from the sequences $S_1 \ldots S_t$ are stored. Sub-sequences are obtained by considering a window of size k and shifting the window by one from left to right till all the sub-sequences are collected.

Although the k-mer segmentation step divides DNA sequences into fixed-length subsequences, the spatial relationship between both adjacent and non-adjacent segments is preserved throughout the proposed framework. A sliding window method is used to carry out the segmentation process in such a way that the original k-mer sequential order is preserved. Also, the positional data of every k-mer is stored and utilized subsequently in the motif joining and extension phases wherein overlapping and adjacent k-mers are joined together to create longer potential motifs. Also, the CNN-based classification is implicitly trained on spatial dependencies between distant segments. As the whole DNA sub-sequence (encoded as a sequence of ordered binary codes) is processed by the convolutional layers, the learned filters can detect not only short patterns of motifs but also longer-range spatial correlation between non-adjacent regions. This hierarchy characteristic learning helps the suggested model to retain and make use of positional circumstances, notwithstanding the initial division phase.

5.3 Popular k-mer Mining

In this step, the most popular k-mers subsequences and their positions will be found by intersect all k-mers subsequence (in all sequences) and Union their positions. This intersection is done by intersect the first two sequences, the results will be intersected with the third sequence, the result will be intersect with the fourth sequence and so on. Finally, we will obtain the most popular k-mers subsequences with their positions which exist in all or most sequences.

5.4 Motif Joining and Extension

This step goes through two important stages.

5.4.1 Joining and Generating the Maximum Length Pattern

This step aims to combine k-mer subsequences to form longer, frequent DNA patterns. A hash table is used to group subsequences by their starting nucleotide (A, T, C, or G), making it easier to locate potential matches. The algorithm takes a list of frequent patterns of length n and produces merged patterns of length $n + 1$.

Two patterns can be joined if the last $n - 1$ characters of the first match the first $n - 1$ characters of the second, and if their positions occur consecutively. When both conditions are met, the final character of the second pattern is appended to form a new candidate pattern. All valid merged patterns are collected. For example, to extend the pattern CGAT (length 4), the algorithm searches for a matching pattern beginning with G using the hash table. Example 1 illustrates this algorithm:

Example 1:

```
p1 = "ATC" with position = 70,
p2 = "TCG" with position = 71
Last n-1 = 2 of p1 = "TC"
First n-1 = 2 of p2 = "TC" → match,
the merge_pattern = "ATCG"
```

5.4.2 Extending the Most Popular Substring (Maximum Pattern)

In this step, we extend each occurrence of most In this step, each occurrence of the most popular subsequences obtained from the previous stage is extended in three possible directions: left, right, and center. The extension procedure is implemented according to the pseudocode presented in Listing 1. The input of the algorithm consists of the maximum-length pattern and its corresponding positions (with length k'), while the output is the extended pattern with length l generated through left, right, and center expansion strategies. This function extends as follows:

1) Left Extension. Extends the motif to the left by adding as many bases as needed (up to the desired length) without going before the sequence start.
2) Right Extension. Extends the motif to the right, ensuring it does not exceed the sequence end.
3) Center Extension. Extends the motif equally on both sides (left and right) as much as possible, balancing the added bases within sequence boundaries.

If the desired length l is shorter than the motif length, an error is raised.

```
Input: Maximum length pattern with their
positions (with length k').
Output: Extended pattern to left, right,
center direction (with length l).
Function ExtendMotifThreeWays
(DNA_sequence, motif_start, motif_end,l):
Input:
DNA_sequence : full DNA sequence (string)
motif_start  : start index of motif
               (0-based)
motif_end    : end index of motif
               (exclusive)
l            : desired total length after
extension
Output:
left_extended, right_extended,
center_extended:
    k' = motif_end - motif_start
    extension_length = l - k'

    If extension_length <  0:
        Raise Error("l must be >= motif
        length")

    // Left Extension
    left_start = Max(0, motif_start -
                 extension_length)
    left_end = motif_end
    left_extended =
    Substring(DNA_sequence, left_start,
              left_end)

    // Right Extension
    right_start = motif_start
    right_end = Min(Length(DNA_sequence),
    motif_end +  extension_length)
    right_extended =
    Substring(DNA_sequence,
    right_start, right_end)

    // Center Extension
    left_ext_len = Floor(extension_length
                 / 2)
    right_ext_len = extension_length -
                    left_ext_len
    center_start = Max(0, motif_start -
                 left_ext_len)
    center_end =
    Min(Length(DNA_sequence),
    motif_end + right_ext_len)
    center_extended =
    Substring(DNA_sequence,
    center_start, center_end)

    Return left_extended, right_extended,
           center_extended
```

Listing 1: Pseudocode for candidate motif extension in three directions (left, right, and center).

5.5 Consensus Motif Mining

This step is divided into four sub steps in order to discover the candidate planted motif.

5.5.1 Collecting

This step will collect the subsequences of each side into separate place, such that the subsequences of left side (for all sequences) are stored in separate place (list or array), also the subsequences of right side (for all sequences) are stored in separate place and the same thing with the center.

5.5.2 Mining The Consensus Motif (Cm) For Each Side

Consensus string is the specific representation of the motif. In motifs with consensus string, characters in each position have the highest frequency in the corresponding column of each motif instance.

This mining process is performed by calculating the profile matrix P for each side of the motif. The consensus motif is then generated by identifying the most frequent nucleotide at each position across aligned motif instances. The procedure for generating the consensus sequence is presented in Listing 2. The function takes a set of equal-length DNA sequences as input and produces a representative consensus sequence by analyzing nucleotide frequencies at each position. It iterates through each column of the aligned sequences, counts occurrences of A, C, G, and T, and selects the nucleotide with the highest

count. If no nucleotide has a clear majority, 'N' is used as a placeholder. The selected nucleotides across all positions form the consensus motif.

```
Input: sequences: List of DNA sequences
(all of equal length).
Output: consensus: A single DNA sequence
representing the consensus motif.
Function GenerateConsensus (sequences)
    sequence_length =
    Length(sequences[0])
    consensus = " "
  For position from 0 to
       sequence_length - 1
       counts = Dictionary with keys
       'A', 'C', 'G', 'T' all
       initialized to 0
       For each seq in sequences
          base = seq[position]
  If base in counts Then
    counts[base] = counts[base] + 1
    max_base = 'N'  # default if no
                      clear winner

// Find the base with the highest
   count
    max_count = 0
    For base in ['A', 'C', 'G', 'T']
    If counts[base] > max_count Then
       max_base = base
       max_count = counts[base]
       consensus = consensus +
                   max_base

   Return consensus
```

Listing 2: Procedure for generating a consensus motif from aligned DNA sequences.

5.5.3 Calculating the Strength Scoring for each Consensus Motif (cm)

This step computes the motif Strength Score MSS of cm by the scoring function explained in equation (1) to compare different guesses and choose the "best" one.

$$\text{Score (s, DNA)} = \sum_{i=1}^{l} \max count\ (k, i), \quad where\ k\ \in (A, C, G, T) \tag{1}$$

Assume s = (s1, s2,…….st) represents t DNA sequences as shown in Table 1. This figure illustrates the calculating of MSS for 5 aligned DNA sequences.

5.5.4 Taking The Upper Scoring Side

By calculating the motif strength score MSS for three sides, we will take the upper scoring side which it will represent the candidate motif.

5.6 Generating of d-neighbors

In computational biology, the motifs may vary slightly in different sequences due to mutations, sequencing errors, or natural biological variation. To account for this, researchers use the concept of d-neighbors. The concept as follows: For a DNA motif of length l and a non-negative integer d, the d-neighbors are all sequences of the same length that differ from the motif in at most d positions, as measured by the Hamming distance. For example, if the motif is ACGT and $d = 1$, its d-neighbors include sequences like CCGT, AAGT, and ACCT, each differing by one nucleotide.

Table 1: Consensus mining and scoring calculating.

	A	T	G	A	A	C	T	T
	T	C	G	A	A	C	G	T
	G	C	G	A	G	A	G	T
	A	C	T	A	G	C	A	T
	A	C	G	A	A	C	G	G
A	3	0	0	5	3	1	1	0
C	0	4	0	0	0	4	0	0
G	1	0	4	0	2	0	3	1
T	1	1	1	0	0	0	1	4
Consensus	A	C	G	A	A	C	G	T
Scoring	3+4+4+5+3+4+3+4 = 30							

5.6.1 Hamming Distance

Definition 1: Let $a = a_1\ a_2 \cdots a_n$ and $b = b_1\ b_2 \cdots b_n$ be two strings from $\Sigma+$. The Hamming distance d (a, b) between a and b is defined as:

$d(a,b) = \sum_{i=1}^{n} \in (ai,bi) = 1$ if $ai \neq bi$, 0 otherwise. It can be interpreted as the number of symbol mutations needed to turn one string into another.

5.6.2 Computational Consideration

Generating d-neighbors is computationally intensive because the number of possible sequences increases exponentially with motif length and the value of d. For a motif of length l, equation (2) shows the count of d-neighbors.

$$N\ (\sum, l, d) = \sum_{i=0}^{d} \binom{l}{i} (|\sum| - 1)^i. \tag{2}$$

Where $\sum$ is the count of DNA symbols which is 4. For example, when l = 14, d = 4 for DNA (where alphabet size = 4), the number of d-neighbors of an l-mer (all strings within Hamming distance ≤ d) is

$\sum_{i=0}^{d} \binom{l}{i}(4-1)^i = .$
$\sum_{i=0}^{4} \binom{14}{i}(3)^i = 1 + 42 + 819 + 9{,}828 + 81{,}081 =$ 91,771.

The number of all d-mutated copies of candidate motif is at most:

$$N(l, d) = \binom{l}{d}(|\sum| - 1)^d. \quad (3)$$

The probability that a randomly chosen l-mer is a d-neighbors of M is:

$$P(\sum, l, d) = P1 = \frac{N(\sum, l, d)}{|\Sigma|^l}. \quad (4)$$

The probability that in a sequence of length m, there is at least one string u that is a d-neighbor of M is:

$$P(m, \sum, l, d) = P2 = 1 - (1 - P1)^{m-l+1}. \quad (5)$$

5.6.3 Importance of D-Neighbors Generation

The generation of d-neighbors is a key step in motif-finding algorithms, enabling the detection of approximate rather than exact motif matches. Since real motifs may contain variations, algorithms generate all possible sequences that differ from a candidate motif by d substitutions. The procedure for generating d-neighbors is presented in Listing 3. The generate_neighbors_with_d_mutations function takes a DNA motif as input and produces all possible sequences that differ from the original motif by exactly d substitutions. The process selects combinations of mutation positions and replaces the original nucleotides with alternative bases, generating a set of unique d-neighbor sequences.

5.7 CNN Model For Motif Classification

This step divided into two sub steps.

5.7.1 Training

Neural network training begins with labeled data pairs (X_i,Y_i), where each input X_i is evaluated against its true label Y_i to compute a loss. Since DNA sequences are one-dimensional, one-dimensional CNNs were used. Various models were trained with different hyperparameters such as the number of layers, filters (64 and 164), and filter window sizes (3, 5, and 7) to determine the optimal configuration. The window size of 3 achieved the highest accuracy, and a batch size of 512 was chosen. The best-performing model was then trained on the full dataset to build the final CNN model.

Each CNN model includes six layers: a convolutional layer with 64 filters (window size 3), followed by a max-pooling layer (pool size 2); a second convolutional layer with 128 filters (window size 3), followed by another max-pooling layer (pool size 2); a dropout layer with a 50% rate to prevent overfitting; a flattening step; a fully connected layer with 64 neurons; and a final softmax output layer for gene probability prediction. The complete procedure for motif dataset generation and CNN model training is presented in Listing 4. The procedure includes synthetic DNA sequence generation, motif insertion, binary encoding, dataset splitting, CNN architecture definition, and model training.

```
Input: Candidate motif(s)
Output: All d-neighbors for candidate
motif(s)
Define function
generate_neighbors_with_d_mutations
(sequence, d):
 Define list nucleotides = ['A', 'T',
                            'C', 'G']
 Initialize empty set neighbors
 Define inner function
 generate_substitutions(seq,
                        positions):
 For each combination of
    substitutions (subs) of
    nucleotides at the given
    positions:
    Convert seq to a list (mutable)
    For each position and
    corresponding
    substitution:
    If the nucleotide at that
    position is not equal to the
    substitution:
       Replace the nucleotide with
       The substitution
       Add the modified sequence (as
       string) to the neighbors set
       Get the length of the input
       sequence (seq_length)
    For each combination of d
    positions from the
    sequence: Call
    generate_substitutions(sequence,
    positions)
 Return the neighbors set
```

Listing 3: Procedure for generating d-neighborhoods with d mutations.

The choice of a one-dimensional convolutional neural network (1D-CNN) is motivated by the intrinsic characteristics of the Planted Motif Problem (PMP), which primarily involves the identification of short, locally conserved patterns embedded within longer DNA sequences. Unlike natural language tasks

that require modeling long-range semantic dependencies, PMP focuses on detecting position-invariant local motifs that may appear at arbitrary locations with limited mutations. CNNs are mathematically designed for translation invariance meaning they are highly effective at detecting specific local patterns regardless of their position. While Transformers excel at global dependencies, the PMP is a 'needle in a haystack' problem where the 'needle' (the motif) is structurally independent of the distant 'hay' (the rest of the sequence).

```
Input: All d-neighbors of candidate motif(s)
Output: Training model on all d-neighbors
1. Load Excel file containing motif d-
neighbors from specified file path
2. Define DNA alphabet: ['A','C','G',T']
3. Define function to generate random DNA
    sequence of given length
4. Define function to insert a given motif
into a random position in a sequence
   - Return modified sequence and label=1
5. Define function to generate dataset:
   - For each motif in d-neighbor list:
       - Generate random DNA sequence
       - With 50% probability, insert the
          Motif (label=1)
       - Else, keep original sequence
         (label=0)
       - Append sequence and label to
         lists
   - Return sequences and labels as
     arrays
6. Define function to binary encode DNA
   sequences:
   - Map A, C, G, T to binary vectors
   - Convert all sequences to corresponding
     Binary format
7. Generate dataset using above functions
   - Two bit encode sequences
8. Split dataset into training and
   testing
   sets (80/20 split)
9. Define a Convolutional Neural Network
   (CNN) with:
   - 2 convolution + pooling layers
   - 1 fully connected layer
   - 1 output sigmoid layer for binary
      classification
10. Compile the model with Adam optimizer
    and binary crossentropy loss
11. Train the model on training data
    (e.g., 3 epochs, batch size 32)
12. Fine-tune hyperparameters and retrain
    If necessary.
13. Save the trained model for future
     predictions.
```

Listing 4: Procedure for synthetic motif dataset generation and CNN model training.

5.7.2 Testing

In the testing step, all subsequences with length l will be extracted from the original dataset D and saved into list, after that this list enters as input to the PMM. The miner will predict if this subsequence (included in the list) is planted motif or not depending on the obtained probability. PThe prediction procedure for identifying planted motifs within the test sequences is presented in Listing 5. The method extracts all subsequences of length l, encodes them using the two-bit representation, and applies the trained CNN model to estimate the probability of motif presence.

Input: Training model + Original dataset
Output: Planted motif prediction

```
For each sequence S in all sequences:
  For each position i from 0 to
  (length of S - l + 1):
    si = substring of S from position
         i to i + l
    Save si into subsequences list
    encoded =
    two_bit_encode(subsequences)
    predictions =
    model.predict(encoded)
    If predictions >
    probability_threshold:
       Add (sequence_number, si,
       position i) to
       planted_motif_list
    Else:
       Discard si
```

Listing 5: Procedure for planted motif prediction using the trained CNN model.

Figure 3 shows a deep learning architecture for DNA sequence classification using a Convolutional Neural Network (CNN) as explained in the following steps:

1) Input (DNA sequence):
   - The DNA sequence (e.g., GGCAATACCT) is first transformed into a numerical format.
   - This is done through binary encoding, where each nucleotide (A, C, G, T) is represented as a binary vector.
2) Convolution and Pooling Layers:
   - The binary encoded sequence is passed through multiple convolutional layers, which act like filters scanning for short patterns (motifs) within the DNA.

- Pooling layers then reduce dimensionality while preserving the most important features.

3) Fully Connected Layer:
   - The extracted features are flattened and fed into fully connected neurons.
   - This stage integrates motif signals across the sequence to detect higher-level biological patterns.
4) Output Layer:
   - The final layer outputs gene probability
   - (classification score), indicating the likelihood that the input sequence belongs to a pecific lass (e.g., binding site, regulatory region, etc.).

6 RESULTS OF THE PROPOSED MODEL

The PMM methods is tested on both simulated and real datasets. Experimental results of this model was obtained on a google Colab pro environment. Table 2 summarizes the parameters and training and testing sample sizes used in the CNN-based motif classification.

To conduct comprehensive testing, 23 simulated datasets were created with varying motif lengths *(l, d)* from (9, 2) to (30, 10). Each dataset contained 10 sample files, each with 20 DNA sequences of length 600, generated using equal probabilities of {A, C, G, T}. A planted motif *M* of length *l* was inserted into random sequence positions after introducing up to *d* mutations. The goal was to recover these motifs and their locations.

The testing process included several steps: data preprocessing, k-mer segmentation, popular k-mer mining, motif joining and extension, consensus motif mining, and *d*-neighbor generation, followed by CNN-based motif classification using Keras and TensorFlow. Table 3 presents the performance results including runtime, training time, accuracy, F1-score, false positive rate (FPR) and loss for selected (l, d) configurations with 20 * 600 bp DNA.

6.1 Experimental Result for Simulated Data

While accuracy provides an overall performance indicator, it may not fully reflect discriminative capability in motif discovery, particularly under challenging ((l,d)) configurations where the mutation rate is high (i.e., the so-called “twilight zone”). In such cases, the number of potential false positives increases substantially. To address this, we evaluated the model using additional metrics including False Positive Rate (FPR) and F1-score.

Table 2: Model parameters.

Parameter	Description
DNA sequence length	600 bp
Motif length	l
Mutation level	d
Training data source	d-neighbors of candidate motif (after sampling, if applied)
Training samples (positive)	One DNA sequence per d-neighbor with motif embedded
Training samples (negative)	One DNA sequence per d-neighbor without motif
Total training samples	2 × (number of d-neighbors after sampling)
Training / Testing split	80% training, 20% validation
Test samples per DNA sequence	$(600 - l + 1)$ subsequences
Total test samples	$t \times (600 - l + 1)$

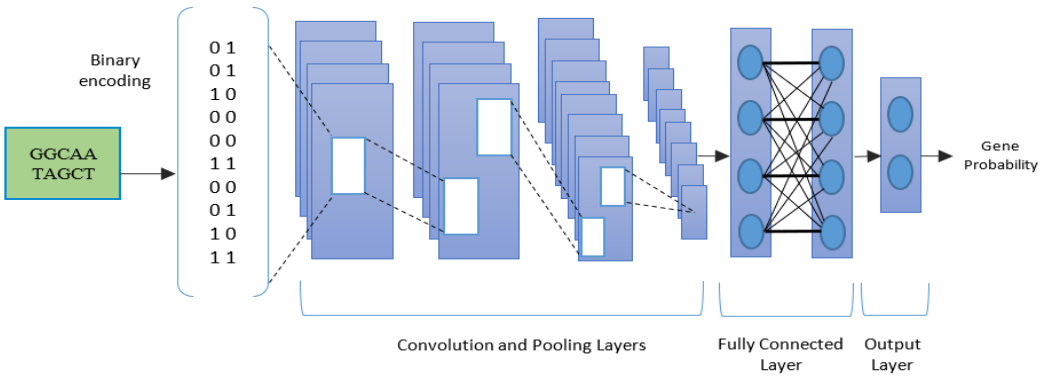

Figure 3: Deep learning architecture for DNA sequence classification.

Table 3: PMM performance.

(l, d)	Count of d-neighbors	Running Time for generating d-neighbors in Colab Pro	No. of epoch	Running time for training in Colab Pro	Accuracy	F1_score	FPR	Loss
(10, 4)	20 686	0.05 Sec	5	5 Sec	0.9881	0.9855	0.0052	0.0772
			10	8 Sec	0.9882	0.9882	0.0044	0.0607
			15	12 Sec	0.9830	0.9831	0.0032	0.0804
			20	15 Sec	0.9909	0.9911	0.0007	0.0547
			25	18 Sec	0.9993	0.9971	0.0004	0.0442
(14, 4)	91 771	0.1 Sec	5	4 sec	0.9723	0.9547	0.0052	0.0772
			10	9 Sec	0.9827	0.9832	0.0030	0.0756
			15	12 Sec	0.9830	0.9831	0.0034	0.0804
			20	16 Sec	0.9877	0.9879	0.0020	0.0743
			25	20 Sec	0.9991	0.9998	0.0014	0.0644
(18, 6)	15 886 504	1 Min	2	6 Min, 35 Sec	0.9995	0.9992	0.0088	0.0015
			4	11 Min	0.9998	0.9995	0.0082	0.0016
			6	15 Min, 22 Sec	0.9999	0.9995	0.0075	0.0012
			8	19 Min, 35 Sec	0.9999	0.9997	0.0077	0.0009
			10	23 Min, 42 Sec	1.0000	0.9997	0.0072	0.0002
(26, 6)	185 106 364	12 Min, 23 Sec	1	53 Min, 53 Sec	0.9990	0.8500	0.0155	0.0022
			2	1 Hour, 21 Min, 8 Sec	0.9995	0.8828	0.0132	0.0020
			3	1 Hour, 50 Min, 15 Sec	0.9998	0.9264	0.0115	0.0005
			4	2 Hour, 17 Min	1.0000	0.9527	0.0094	0.0032
			5	2 Hour, 44 Min, 7 Sec	1.0000	0.9788	0.0090	0.0027

The simulated data was done on a uniform background model to ensure comparison with standard PMP benchmarks. Nonetheless, PMM is not based on background uniformity and its strength can be proved by experiments on actual genomic data that naturally obey complex and non-uniform (Markovian) distributions. Moreover, the CNN-based classifier learns discriminating against noise of high-entropy. Further research will explicitly test PMM on backgrounds of higher order Markov.

6.1.1 Overcoming Computational Challenges in Large Motif Mining Tasks

A length of planted motif length (l) greater than 26 characters and a mutation level (d) greater than 6 results in the formation of a vast number of d-neighbors, ranging between millions and billions. Planted motif mining and model training are therefore very costly. In order to get over this difficulty, we employed the following methods to secure satisfactory accuracy:

1) Random sampling. In the proposed PMM framework, random sampling is introduced only when the number of d-neighbors becomes computationally infeasible (on the order of 10^{12}–10^{15}). The sampling strategy is used as a computational approximation, not as a statistical estimator of motif frequency. The goal of sampling in PMM is not to estimate the complete d-neighbor space, but to provide a sufficiently diverse training set for the CNN classifier. The CNN's task is to learn a decision boundary between motif-like and non-motif sequences, rather than to exhaustively enumerate all d-neighbors.
2) Multiprocessing strategy. We adopted a multiprocessing solution in order to scaleably compute d-neighbors. The parallelization allows reducing the runtime of the code considerably in comparison with the sequential methods. In order to maximize memory and I/O efficiency, the study made use of batch processing and chunked file writing and experimentation was conducted on the Google Colad platform.
3) Managing Disk Usage in Colab. To reduce the disk overflow problems due to the small storage capacity of Google Colab.

When creating more than 1.7 trillion d neighbors, we used a scaling approach:

1) Random Sampling and Thresholding. In case of excessively large generation of neighbors, we used random sampling to control data volume.
2) Chunked Storage & Offloading. Subsets of neighbors (i-neighbors), where i were written in chunks directly to Google Drive.
3) Space Reclamation. Once successfully offloaded to persistent cloud storage, these files were removed from the local Colab environment to conserve disk space.
4) Table 4 presents performance metrics for generating and training d-neighbors of DNA sequences across different motif lengths (l) and mutation levels (d), including neighbor counts (before and after sampling), time1 represents time of generating d-neighbors with multiprocessing but without saving into google drive, time2 represents time of generating d-neighbors with multiprocessing and saving into google drive. Generation and training times using multiprocessing and Google Drive storage under parameters: 10 workers, batch size 100,000, chunk size 1,000,000, and 5 epochs.
5) **6.1.2 Testing**
6) After training the CNN model, the testing phase was
7) conducted to evaluate the model's efficiency and computational performance on various planted motif configurations. The testing process aimed to measure how well the model could identify motifs within artificially generated DNA sequences containing planted motifs of different lengths (l) and mutation levels (d).
8) Table 5 presents the planted motifs from determined simulated dataset (which exist in NA_30_12 sample_3) for candidate motif (CTCAGGGTCGACTCGAAGGAACAG CTAACT) where l = 30, d = 12. For every sequence, the most similar motif pattern is shown along with its position(s) within the sequence and the prediction score with corresponding similarity scoring. These scores are calculated according to (6).

*Similarity Scoring = (number of matched characters in the taken subsequence / l) * 100* (6)

Table 4: Computational performance for d-Neighbor and Model Training across large difference (1,d).

(l, d)	d-neighbors count (before sampling)	d-neighbors count (after sampling)	Time1	Time2	CNN Results				
					Training time	Accuracy	F1 score	FPR	Loss
(18, 6)	15,883,504	15 883 504 (0 %)	3 Sec	36 Sec	6 Min, 36 Sec	0.9998	0.9993	0.0082	0.0013
(26, 6)	185,090,364	185 090 364 (0 %)	36 Sec	9 Min, 23 Sec	2 Hour, 44 Min, 7 Sec	1.0000	0.9788	0.0090	0.0000
(28, 6)	300 275 914	150 137 957 (50 %)	32 Sec	7 Min, 10 Sec	1 Hour, 20 Min, 20 Sec	0.9985	0.9488	0.0132	0.0765
(28, 8)	23 282 048 299	465 640 965 (2 %)	2 Min, 25 Sec	22 Min 13 Sec	5 Hour, 31 Min, 33 Sec	0.9500	0.9357	0.0313	0.0562
(30, 6)	469 757 864	234 878 932 (50 %)	55 Sec	11 Min, 17 Sec	2 Hour, 46 Min, 33 Sec	0.9782	0.9594	0.0553	0.0587
(30, 7)	4 920 897 464	246 044 873 (5 %)	1 Min 22 Sec	12 Min, 9 Sec	3 Hour, 18 Min, 40 Sec	0.9280	0.9444	0.0587	0.0562
(30, 8)	43 329 616 389	433 296 163 (1 %)	2 Min 12 Sec	20 Min, 42 Min	5 Hour, 12 Min, 52 Sec	0.9306	0.8844	0.0654	0.0762
(30, 9)	324 896 524 339	324 896 524 (0.1 %)	1 Min 32 Sec	31 Min, 22 Sec	4 Hour, 12 Min, 52 Sec	0.8923	0.8528	0.0768	0.3762
(30, 10)	2 102 825 570 034	630 847 671 (0.003 %)	2 Min 51 Sec	58 Min	7 Hour, 33 Min, 52 Sec	0.8753	0.8246	0.0944	0.4762

6.1.3 Comparison with other Algorithms

Table 6 shows the running time comparison between our proposed model (PMM) and four existing motif discovery algorithms-qPMS10, FMotif, GADEM, and MEME-ChIP-across a range of challenging instances (l, d) from (9, 2) to (30, 10).

The results demonstrate that PMM consistently achieves faster execution times, particularly for larger and more complex instances (the running time includes consensus motif time plus d neighbors time and prediction time). While qPMS10 fails to produce results (denoted as N) for $l \geq 23$ and FMotif, GADEM, and MEME-ChIP require more than 24 hours to complete, PMM completes the same tasks in minutes. These findings highlight the superior computational efficiency and scalability of the proposed PMM compared with traditional motif discovery methods. Where S Abbreviation for seconds, m for minutes, h for hours, N for no results because the running time more than 48 hours, - for not mentioned.

The compared algorithms include qPMS10 [6], FMotif [23], GADEM [24] and MEME-ChIP [25]. FMotif is the most efficient exact qPMS algorithm for processing large DNA datasets, GADEM is designed for motif discovery on large DNA datasets by combining a genetic algorithm and an EM algorithm, MEME-ChIP is one of the most famous motif discovery algorithms. For each setting of t, n, l, d, we evaluate our results based on the results of four other algorithms mentioned earlier.

In addition, we can analyze how the performance of these algorithms is affected by the parameters t, n, l, and d from the experimental results. For GADEM and MEME-ChIP, their running time is mainly affected by the number of sequences t. our model is slightly affected by (l, d). Specifically, as shown in the results, the running time of our model increases with the increase of (l, d). The reason for this phenomenon is that, as (l, d) grows, more d-neighbors need to be verified in the refinement process of the model. FMotif is heavily affected by (l, d), as the number of candidate motifs of FMotif shows exponential growth with the increase of (l, d).

Figure 4 compares the Running Time (seconds) of the Proposed Model (PMM) against the four algorithms (qPMS10, FMotif, GADEM, and MEME-ChIP) across various input sizes, denoted as (l, d). The graph demonstrates that PMM (blue line) significantly outperforms the other algorithms in terms of efficiency, particularly as the input size increases.

Table 5: Patterns found and associated with scoring for (CTCAGGGT CGACTCGAAGGAACAGCTAACT).

Seq no.	Found Pattern	Running Time	Positions	Prediction Score	Similarity Scoring
1	CTCAGGGTGGATTCGAACGTACAGTTAACG	3.3 Sec	268	6	80 %
2	CTCAAGATCGACTCGATGGTTCCGCTAACG	3.3 Sec	476	7	76.67 %
3	ATGCGGGTCGACTGGGAGGAACCAATTACG	3.3 Sec	352	10	66.67 %
4	CTTTGAGTCGCCTCTAATGAACAACTAACT	3.3 Sec	342	7	76.67 %
5	CGCGGGATGGACTCGAAGAAACAGCTAACT	3.3 Sec	94	5	83.33 %
6	ATCAGGGTCGCCGCAAAGCATCAGCTAACT	3.3 Sec	472	6	80 %
7	CTCATGGTCCGCTGAACTGTACGGCTTGCT	3.3 Sec	175	11	63.33 %
8	CTCAGGGTCGACTCGAAGGAACAGCTAACT	3.3 Sec	69	0	100 %
9	CTCAGGGTCGACTCGAAGGAACAGCTAACT	3.3 Sec	429	0	100%
10	CACAGGGCCGACTGGAATGAGCCGCTAAAT	3.3 Sec	249	7	76.67 %
11	CTAGCCGTAGACTTCAAGCCCCAGCTGACG	3.3 Sec	32	12	60 %
12	CGCTGTGTCGAACCGAAGCAGATGCTAGGT	3.3 Sec	20	11	63.33 %
13	CTCGGGGTCGTCTGGAAGAAACAGCTGACC	3.3 Sec	358	6	80 %
14	CCCCGGGTCTACTCGAACGAAGCGCCAACT	3.3 Sec	264	7	76.67 %
15	GTCAGGGTCGTCTTGAAAGAACCGCAACCT	3.3 Sec	462	7	76.67 %
16	GTCAGGGGCGACTTGAAATAACATCTAACT	3.3 Sec	543	6	80 %
17	TTGGGCGACGAGTCGAAGCAACAACTAACT	3.3 Sec	253	8	73.33 %
18	CTAAGCGCCGATTCGTACGAACAGACGATT	3.3 Sec	218	10	66.67 %
19	AGGACGGTACACTCTAAGGGCCAGCTGACT	3.3 Sec	314	10	66.67 %
20	CGCGGTGCCGATTCGAAGGGACAGCTAACG	3.3 Sec	151	7	76.67 %

Table 6: Results on the first set of simulated data.

(l, d)	Proposed Model (PMM)	qPMS10	FMotif	GADEM	MEME-ChIP
(9, 2)	0.2 + 0.01 + 0.3 = 0.52 sec	-	1.7 m	1.3 h	3.9 h
(11, 3)	0.3 + 0.1 + 0.6 = 1 sec	-	18.7 m	1.4 h	3.9 h
(13, 4)	0.5 + 0.2 + 0.8 = 1.5 sec	-	2.6 h	1.4 h	3.8 h
(15, 5)	1 + 2 + 1 = 4 sec	-	26.4 h	1.4 h	3.7 h
(17, 6)	1 + 42 + 1 = 44 Sec	2.9 m	N	1.3 h	3.6 h
(19, 7)	2 + 8 Min, 40 Sec + 3 = 8 Min, 45 Sec	14.4 m	N	1.3 h	3.6 h
(21, 8)	2 + 12 min, 33 sec + 3 = 12 min, 38 sec	50.2 m	N	1.3 h	3.5 h
(23, 9)	2 + 55 min, 35 sec + 3 = 55 min, 43 sec	2.49 h	N	-	-
(26, 6)	2 + 13 Min, 56 sec + 3 = 14 min, 1 sec	N	N	-	-
(28, 8)	3 + 22 Min, 13 sec + 4 = 22 Min, 20 sec	N	N	-	-
(30, 10)	3 + 58 min + 4 = 58 Min, 7 sec	N	N	-	-

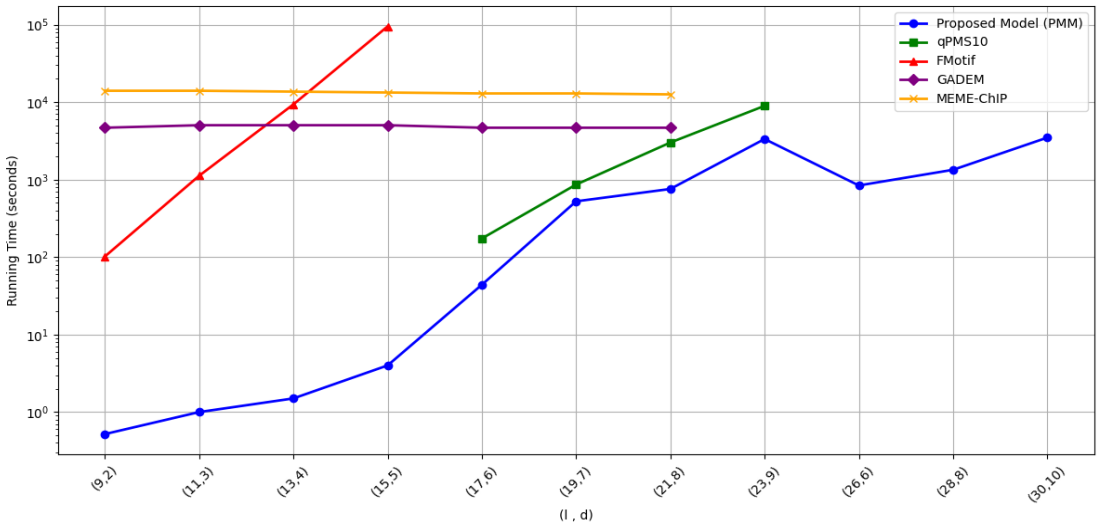

Figure 4: Comparison of running time for planed motif algorithms.

7 CONCLUSIONS

As experiments demonstrate, the proposed Planted Motif Model (PMM) is useful to detect planted motifs in simulated datasets of DNA. It is more accurate, scalable and efficient in computation. PMM is able to obtain almost perfect accuracy (> 0.98) and small values of loss at different lengths of motifs and levels of mutations, which proves its strength in performing motif identification tasks.

With a combination of CNN-based classification and multiprocessing and random sampling, the model is able to handle the exponent of d-neighbors in large (l, d)-configurations. These optimizations enable efficient training and inference of large data-sets, including (30, 10) with over 2 trillion d-neighbors, reducing the requirement of large amounts of runtime and storage, as well as the prediction accuracy.

The paper shows that our model is very competitive compared to state of art methods and Experimental results also reveal that our algorithm performs better especially when the dataset is large.

PMP with at most d mutations is much harder than that with exact d mutations since the former has much larger search space than the latter.

The set of potential variations of the motif (d-neighbors) grows exponentially to millions or billions of potential variations in motifs longer than 26 characters and with more than 6 mutations. This poses a serious computational bottleneck, and therefore the mining process and model training is extremely burdensome.

8 FUTURE WORK

The following suggestions can be recommended for future works.

1) Parallelization for Efficiency. The process of generating d-neighbors and training the model will be parallelized using multi-core CPUs and GPUs to reduce runtime and handle larger genomic datasets more efficiently.
2) Evaluation on Real-World Datasets. The model will be tested on diverse biological datasets from different organisms and genomic regions to ensure its accuracy, robustness, and real-world applicability;
3) Hybrid CNN–RNN Architectures. Upcoming iterations will integrate CNNs with RNN architectures, such as LSTM or GRU, to simultaneously identify local motif features and long-range sequence relationships, thereby enhancing overall detection performance.

REFERENCES

[1] V. V. S. Dileep, R. Navuduru, R. Gummadi, and P. Natarajan, "DNA sequencing using machine learning and deep learning algorithms," International Journal of Innovative Technology and Exploring Engineering (IJITEE), vol. 11, no. 10, 2022.

[2] D. Sharma and S. Rajasekaran, "A simple algorithm for (l, d) motif search," Search, pp. 148-154, 2009.

[3] C. Pizzi, "Motif discovery with compact approaches: Design and applications," IntechOpen, 2011.

[4] Q. Yu, H. Huo, X. Chen, H. Guo, J. S. Vitter, and J. Huan, "An efficient motif finding algorithm for large DNA data sets," in IEEE International Conference on Bioinformatics and Biomedicine (BIBM), 2014.

[5] J. Davila, S. Balla, and S. Rajasekaran, "Space and time efficient algorithms for planted motif search," in V. N. Alexandrov et al., Eds., ICCS 2006, Part II, Lecture Notes in Computer Science, vol. 3992, pp. 822-829, Springer, 2006.

[6] P. Xiao, S. Pal, and S. Rajasekaran, "qPMS10: A randomized algorithm for efficiently solving quorum planted motif search problem," in IEEE International Conference on Bioinformatics and Biomedicine (BIBM), USA, 2016.

[7] D. Quang and X. Xie, "DanQ: A hybrid convolutional and recurrent deep neural network for quantifying the function of DNA sequences," Nucleic Acids Research, vol. 44, p. e107, 2016.

[8] Z. Shen, W. Bao, and D.-S. Huang, "Recurrent neural network for predicting transcription factor binding sites," Scientific Reports, vol. 8, pp. 1-10, 2018.

[9] X. Pan, P. Rijnbeek, J. Yan, et al., "Prediction of RNA-protein sequence and structure binding preferences using deep convolutional and recurrent neural networks," BMC Genomics, vol. 19, p. 511, 2018.

[10] C. Angermueller, T. Pärnamaa, L. Parts, and O. Stegle, "Deep learning for computational biology," Molecular Systems Biology, vol. 12, no. 7, p. 878, 2018.

[11] S. Min, B. Lee, and S. Yoon, "Deep learning in bioinformatics," Briefings in Bioinformatics, vol. 18, no. 5, pp. 851-869, 2017.

[12] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning, MIT Press, 2016.

[13] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature, vol. 521, no. 7553, p. 436, 2015.

[14] M. A. Di Gangi, S. Gaglio, C. La Bua, G. L. Bosco, and R. Rizzo, "A deep learning network for exploiting positional information in nucleosome-related sequences," in International Conference on Bioinformatics and Biomedical Engineering, pp. 524-533, Springer, 2017.

[15] D. Asir, S. Appavu, and E. Jebamalar, "Literature review on feature selection methods for high-dimensional data," International Journal of Computer Applications, vol. 136, no. 1, pp. 9-17, 2016.

[16] Y. He, Z. Shen, Q. Zhang, S. Wang, and D.-S. Huang, "A survey on deep learning in DNA/RNA motif mining," Briefings in Bioinformatics, vol. 22, no. 4, pp. 1-10, 2021.

[17] S. Wang and T. Huang, "Applications of deep learning in biomedicine," in Reference Module in Biomedical Sciences, pp. 1-11, Elsevier, 2019.

[18] Y. Li, C. Huang, L. Ding, et al., "Deep learning in bioinformatics: Introduction, application, and perspective in the big data era," Methods, vol. 166, pp. 4-21, 2019.

[19] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, "Convolutional neural networks: An overview and application in radiology," Insights into Imaging, vol. 9, pp. 611-629, 2018.

[20] N. Tajbakhsh, J. Y. Shin, S. R. Gurudu, et al., "Convolutional neural networks for medical image analysis: Full training or fine tuning," IEEE Transactions on Medical Imaging, vol. 35, pp. 1299-1312, 2016.

[21] Q. Li, W. Cai, X. Wang, et al., "Medical image classification with convolutional neural networks," in 13th International Conference on Control Automation Robotics & Vision (ICARCV), pp. 844-848, 2014.

[22] S. H. S. Basha, S. R. Dubey, V. Pulabaigari, and S. Mukherjee, "Impact of fully connected layers on performance of convolutional neural networks for image classification," Neurocomputing, vol. 378, pp. 112-119, 2020.

[23] C. Jia, M. B. Carson, Y. Wang, Y. Lin, and H. Lu, "A new exhaustive method and strategy for finding motifs in ChIP-enriched regions," PLoS ONE, vol. 9, no. 1, p. e86044, 2014.

[24] L. Li, "GADEM: A genetic algorithm guided formation of spaced dyads coupled with an EM algorithm for motif discovery," Journal of Computational Biology, vol. 16, no. 2, pp. 317-329, 2009.

[25] P. Machanick and T. L. Bailey, "MEME-ChIP: Motif analysis of large DNA datasets," Bioinformatics, vol. 27, no. 12, pp. 1696-1697, 2011.