Fall 12-31-2018

Deep learning methods for mining genomic sequence patterns

Xin Gao
New Jersey Institute of Technology

Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

DEEP LEARNING METHODS FOR MINING GENOMIC SEQUENCE PATTERNS

by
Xin Gao

Nowadays, with the growing availability of large-scale genomic datasets and advanced computational techniques, more and more data-driven computational methods have been developed to analyze genomic data and help to solve incompletely understood biological problems. Among them, deep learning methods, have been proposed to automatically learn and recognize the functional activity of DNA sequences from genomics data. Techniques for efficient mining genomic sequence pattern will help to improve our understanding of gene regulation, and thus accelerate our progress toward using personal genomes in medicine.

This dissertation focuses on the development of deep learning methods for mining genomic sequences. First, we compare the performance between deep learning models and traditional machine learning methods in recognizing various genomic sequence patterns. Through extensive experiments on both simulated data and real genomic sequence data, we demonstrate that an appropriate deep learning model can be generally made for successfully recognizing various genomic sequence patterns. Next, we develop deep learning methods to help solve two specific biological problems, (1) inference of polyadenylation code and (2) tRNA gene detection and functional prediction. Polyadenylation is a pervasive mechanism that has been used by Eukaryotes for regulating mRNA transcription, localization, and translation efficiency. Polyadenylation signals in the plant are particularly noisy and challenging to decipher. A deep convolutional neural network approach DeepPolyA is proposed to predict poly(A) site from the plant Arabidopsis thaliana genomic sequences. It employs various deep neural network architectures and demonstrates

its superiority in comparison with competing methods, including classical machine learning algorithms and several popular deep learning models. Transfer RNAs (tRNAs) represent a highly complex class of genes and play a central role in protein translation. There remains a de facto tool, tRNAscan-SE, for identifying tRNA genes encoded in genomes. Despite its popularity and success, tRNAscan-SE is still not powerful enough to separate tRNAs from pseudo-tRNAs, and a significant number of false positives can be output as a result. To address this issue, tRNA-DL, a hybrid combination of convolutional neural network and recurrent neural network approach is proposed. It is shown that the proposed method can help to reduce the false positive rate of the state-of-art tRNA prediction tool tRNAscan-SE substantially. Coupled with tRNAscan-SE, tRNA-DL can serve as a useful complementary tool for tRNA annotation. Taken together, the experiments and applications demonstrate the superiority of deep learning in automatic feature generation for characterizing genomic sequence patterns.

DEEP LEARNING METHODS FOR MINING GENOMIC SEQUENCE PATTERNS

by
Xin Gao

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Computer Science

Department of Computer Science

December 2018

APPROVAL PAGE

DEEP LEARNING METHODS FOR MINING GENOMIC SEQUENCE PATTERNS

Xin Gao

Dr. Zhi Wei, Dissertation Advisor — Date
Associate Professor of Computer Science, New Jersey Institute of Technology

---

Dr. Vincent Oria, Committee Member — Date
Professor of Computer Science, New Jersey Institute of Technology

---

Dr. Xiaoning Ding, Committee Member — Date
Assistant Professor of Computer Science, New Jersey Institute of Technology

---

Dr. Hai Nhat Phan, Committee Member — Date
Assistant Professor of Informatics, New Jersey Institute of Technology

---

Dr. Antai Wang, Committee Member — Date
Associate Professor of Mathematical Sciences, New Jersey Institute of Technology

BIOGRAPHICAL SKETCH

Author: Xin Gao

Degree: Doctor of Philosophy

Date: December 2018

Undergraduate and Graduate Education:

- Doctor of Philosophy in Computer Science,
  New Jersey Institute of Technology, Newark, NJ, USA, 2018
- Bachelor of Engineering in Information Security,
  Nankai University, Tianjin, China, 2012
- Bachelor of Laws,
  Nankai University, Tianjin, China, 2012

Major: Computer Science

Publications and Posters:

X. Gao and Z. Wei, "Deep learning approaches for genomic sequence pattern recognition," *(under submission).*

X. Gao, Z. Wei and H. Hakonarson, "tRNA-DL: A deep learning approach to improve tRNAscan-SE prediction results," *in Human Heredity 2018.*

X. Gao, J. Zhang, Z. Wei and H. Hakonarson, "DeepPolyA: a convolutional neural network approach for polyadenylation site prediction," *IEEE Access*, 6, pp.24340-24349, 2018.

X. Gao, J. Zhang and Z. Wei, "Deep Learning for Sequence Pattern Recognition," *15th IEEE International Conference on Networking, Sensing and Control (ICNSC)*, pp. 1-6, Mar 2018.

J. Tian, T. Yan, X. Gao and G. Wang, "Scheduling survivability-heterogeneous sensor networks for critical location surveillance," *ACM Transactions on Sensor Networks (TOSN)*, 11(4), p.56, 2015.

X. Gao, J. Tian and G. Wang, "Poster: detection of transportation mode based on smartphones for reducing distracted driving," *Proceedings of the 20th ACM International conference on Mobile computing and networking (MobiCom)*, pp. 355-358, Sep 2014.

J. Tian, W. Zhang, G. Wang and X. Gao, "2D k-barrier duty-cycle scheduling for intruder detection in wireless sensor networks," *Elsevier Computer Communications (COMCOM)*, 43, pp.31-42, 2014.

X. Gao, J. Tian, X. Liang and G. Wang, "ARPP: An Augmented Reality 3D ping-pong game system on Android mobile platform," *23rd IEEE Wireless and Optical Communication Conference, (WOCC)*, pp.1-6, 2014.

J. Tian, G. Wang, X. Gao and K. Shi, "User behavior based automatical navigation system on android platform," *23rd IEEE Wireless and Optical Communication Conference, (WOCC)*, pp.1-6, 2014.

To my beloved parents: Wantian Gao and Xirong Cao.

ACKNOWLEDGMENT

I would like to take this opportunity to express my inmost and heartfelt gratitude to my advisor, Dr. Zhi Wei. I feel exceedingly fortunate to become his doctoral student and get the advice from him, such as finding research topics, improving my scientific thinking, revising research papers and also improving technical writing skills. Without his warm care and encouragement, I could not imagine how I could have gone through the toughest moments and overcomed difficulties these years. For a student like me who study overseas alone without family nearby, I honestly think him not only as my advisor but also as my family. His mentorship helps me become an independent researcher.

I am extremely grateful to Dr. Vincent Oria, Dr. Xiaoning Ding, Dr. Hai Nhat Phan and Dr. Antai Wang for serving as my committee members. Thanks for their academic advice in scope and out of scope of my research area and their constructive comments on helping me think thoroughly of all borderline cases and improving my experiment performance. I wish to thank Dr. Cristian M. Borcea, Dr. Ali Mili, Dr. Guiling Wang, Dr. David Nassimi, Dr. James Geller, Dr. Usman W. Roshan and Dr. George Olsen for their support and kind help all the time. In addition, I would like to extend special thanks to my collaborators Dr. Hakon Hakonarson from the Department of Pediatrics in University of Pennsylvania School of Medicine, Dr. Tao Liang from NIO car company, Dr. Jingwei Xu from HERE Technologies and Dr. Li Li from Futurewei Technologies. Their support helps me overcome many difficulties.

Finally, I want to thank my lab mates: Dr. Jie Zhang, Fei Tan, Tian Tian and Kuang Du for exchanging research ideas and discussing programming problems with me. I am thankful to all my NJIT friends I made during my graduate studies. Their friendship lights up my life.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF TABLES

Table **Page**

LIST OF FIGURES

LIST OF FIGURES
(Continued)

Figure **Page**

CHAPTER 1

INTRODUCTION

Deep Learning has achieved tremendous success in many fields and applications such as computer vision [1, 2, 3], image analysis [4, 5], and natural language processing [6, 7, 8]. Deep learning has been also applied as a superb way in the genomics field, such as predicting noncoding RNAs [9, 10], and microRNAs [11]. This dissertation investigates the performance of two main deep learning architectures, Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN), in recognizing various genomic sequence patterns. We compare with traditional machine learning methods (Logistic Regression, Support Vector Machine, Decision Tree, and Random Forest), which use the extracted k-mer features as input. In contrast, the proposed deep learning methods can be applied directly to the raw sequence data without involving extensive manual feature engineering. We design several representative sequence patterns for evaluating these methods. For each pattern, we generate a dataset consisting of both positive samples and negative samples, where positive samples are DNA sequences containing the pattern, and negative samples are DNA sequences do not contain the pattern. We further consider two kinds of negative samples: the random format and the misleading format. Experimental results demonstrate that CNN and RNN have comparable performance and the two deep learning approaches usually outperform traditional machine learning methods in sequence pattern recognition.

Next, we develop deep learning methods to help solve two specific biological problems, (1) inference of polyadenylation code and (2) tRNA gene detection and functional prediction. Polyadenylation plays critical roles in gene regulation, especially in the processes such as mRNA metabolism, protein diversification, and protein localization [12]. The polyadenylation site, also called the poly(A) site, is

defined by surrounding RNA segments and conserved across metazoans with some minor variations in mammals [12, 13, 14]. Accurate prediction of poly(A) sites and identification of motifs that controlling them are fundamental for interpreting the patterns of gene expression, improving the accuracy of genome annotation and comprehending the mechanisms that governing gene regulation [15, 16]. Transfer RNA (tRNA) plays a central in protein translation [17]. Thus, accurate prediction and annotation of tRNA are fundamental for interpreting and improving protein translation in all living cells. The most popular computational algorithm for tRNA prediction tRNAscan can be dated back more than 20 years ago [18]. It develops a seminal covariance model to describe the secondary structural profile and primary sequence consensus of an RNA sequence. tRNAscan-SE is the leading tool for tRNA annotation that has been widely used in the field. However, tRNAscan-SE can return a significant number of false positives when applied to large sequences. Recently, conventional machine learning methods have been proposed to address this issue, but their efficiency can be still limited due to their dependency on handcrafted features and domain knowledge.

With the increasing availability of extensive genomic datasets and leading computational techniques, deep learning methods have been applied to automatically identify and understand gene regulation directly from genomic sequences and predict unknown sequence profiles, for example, poly(A) sites prediction or tRNA prediction. The above researches show that we could apply deep learning models for mining genomic sequence patterns without handcrafted features. However, each special genomic sequence pattern needs a special deep learning model for prediction. Thus, it is essential to find out the rules between genomic sequence patterns and deep learning models. Moreover, we could mine more sequence patterns not only from genomic sequences but also text sequences in natural language processing in our future work.

The rest of the dissertation is organized as follows. Chapter 3 focuses on the comparative study of different deep learning approaches for genomic sequence pattern recognition. It designs several representative DNA sequence patterns, and describes the algorithm to generate the datasets with these sequence patterns. We design the convolutional neural network and the recurrent neural network. We compare their performance with several traditional machine learning models based on simulated datasets. Furthermore, we evaluate these methods on a dataset of real gene sequences, from which we find deep learning models perform better than traditional machine learning methods in recognizing sequence patterns. Chapter 4 proposes a deep convolutional neural network method for predicting polyadenylation sites from the plant Arabidopsis thaliana gene sequences. We investigate various deep neural network architectures and evaluate their performance against classical machine learning algorithms and several popular deep learning models. Chapter 5 focuses on the tRNA genomic sequences prediction. It introduces a deep learning approach to efficiently and effectively improve the tRNAscan-SE prediction results. We conclude this dissertation in chapter 6.

CHAPTER 2

BACKGROUND

In this chapter, we start with an introductory approach to genomic sequence and its underlying knowledge. Then, we will briefly review classical machine learning algorithms and lastly introduce deep learning main concepts and the most commonly used deep learning and distributed frameworks.

2.1 Genomic Sequence

DNA stands for Deoxyribonucleic Acid, while RNA stands for Ribonucleic Acid. DNA is known to be the key molecule in every living organism as it carries the genetic information concerning each individual. DNA molecules are formed by two strands that form a double-helix. Those strands are composed of nucleotides. Each nucleotide contains a sugar (deoxyribose), a phosphate group and one nitrogenous base. There are four bases that can be present on DNA: Cytosine, Adenine, Guanine and Thymine. RNA is a molecule responsible for the coding, decoding, regulation and expression of genes [19]. RNA and DNA have a similar structure, however, RNA only has a single strand that folds onto itself and its sugar is Ribose. It is composed of four types of ribonucleotide bases: Adenine, Cytosine, Guanine and Uracil.

The genomic sequences we used in our experiments are all in FASTA format. A FASTA format (https://zhanglab.ccmb.med.umich.edu/FASTA/) file contains text file information concerning nucleotides or peptide sequences. It consists of a description line followed by the correspondent sequence representation. The description line begins with a “>” followed by the name and/or the sequence identifier.

2.2 Machine Learning Classification Algorithms

In this subsection, we will briefly explain some of the commonly used classification algorithms. Logistic Regression (LR) is a statistical method used for analyzing a dataset in which there are one or more independent variables that output a binary outcome. Its goal is to find the best fitting model as a linear combination of the predictor variables.

Support Vector Machines (SVM) are used as a classifier algorithm where a separating hyperplane is defined in order to separate the classes. The hyperplane is optimal when it has the largest distance to the nearest point in both training classes.

K-nearest neighbors algorithm (KNN) is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether it is used for classification or regression. In KNN classification, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors. If k equals to 1, then the object is simply assigned to the class of that single nearest neighbor. In KNN regression, the output is the property value for the object. This value is the average of the values of its k nearest neighbors.

Bayesian Network (BN) is a probabilistic directed acyclic graphical model (a type of statistical model) that represents a set of variables and their conditional dependencies *via* a directed acyclic graph (DAG).

Decision Trees (DT) are an algorithm used for classification, the goal of which is to create a model that can predict the value of a variable given several input variables. Each non-leaf node in decision trees represents input features and each leaf node on the tree represents the resulting value for that variable given the path from the root to the node.

Random Forests (RF) are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes or mean prediction of the individual trees.

Adaptive Boosting (AdaBoost) is a machine learning meta-algorithm that can be used in conjunction with many other types of learning algorithms to improve performance. The output of the other learning algorithms ("weak learners") is combined into a weighted sum that represents the final output of the boosted classifier.

2.3 Deep Neural Networks

Recent studies have demonstrated that deep learning can solve genomic problems in a more accurate way than traditional machine learning approaches. Deep Learning is a subfield of machine learning, so, before being able to understand deep learning it is mandatory to know what is machine learning and how are computers able to learn. Machine learning is a computer science field that tries to give a machine the ability to learn. Deep learning is a buzzword that derives from artificial neural networks. An artificial neural network is an information processing paradigm that was inspired by the way a biological brain works. Deep architectures include many variants, where the two prominent domains are Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs).

CNNs are widely used for computer vision, and RNNs are mainly used for natural language processing [20, 21]. Recent works based on CNNs allowed training directly with the DNA sequences rather than extracting features beforehand [22, 23, 24]. The connections between units inside RNNs can form a directed graph along a sequence, which allowed RNNs to be another effective and efficient way for feature extraction from DNA sequences [25, 26]. A deep convolutional network is a category of multilayer neural networks, which can model non-linear relationships and

are always organized by a sequential layer-by-layer structure executing a sequence of functional transformations [27]. Recent works based on CNN allowed training directly on genomic sequences rather than extracting features beforehand [22, 23, 24]. In CNN, the number of model parameters is significantly decreased compared to fully connected neural networks *via* convolution operation and parameter sharing. In addition, convolutional layers could extract high-level features from the raw sequences, which has the similar function to the traditional position-weight matrices (PWMs) [28]. Neurons in a fully connected layer have full connections to all activations in the previous layer, while neurons in a convolutional layer share parameters in a particular feature map. Sub-sampling or spatial pooling is used to reduce the dimensionality of each feature map while retaining the main information. This step can be done using different types of pooling like Max, Average, Sum and so on. The max-pooling layer is used to summarize the activations of some specific adjacent neurons by their maximum value, which can help reduce the overfitting of the training data for the model. Fully connected layers are utilized at the end of the network after feature extraction and consolidation, which are performed by the convolutional and pooling layers to create final nonlinear combinations of features. Batch size is the number of samples' each time we fetch from the training set. The most relevant motifs are automatically inferred by the deeper layers during model training, and can be visualized and analyzed through heatmaps and sequence logos [29].

Recurrent Neural Networks (RNNs) belong to a class of artificial neural networks that, unlike traditional neural networks, can contain directed cycles. The idea behind RNNs is to make use of sequence data. They are used, for instance, on speech recognition problems, translation, gene expression and so on. These networks are called recurrent because they perform the same task for every element of a sequence, the output of a computation depends on the last computations. The two popular networks based on RNNs for language modeling are Gated recurrent units (GRUs)

and Long short-term memory units (LSTM) [30, 31]. GRU layer and LSTM layer are similar to each other. The difference between them is that a GRU has two gates and an LSTM has three gates. Both architectures yield comparable performance but utilize the distinct methods to prevent vanishing gradient problem. GRUs have no memory unit and fewer parameters, and thus it exposes the full hidden content without any control and may train a bit faster or need fewer data to generalize. LSTMs are very similar to RNN except that the hidden state is calculated. LSTMs solve the vanishing gradient by introducing structures called "gates" that give them the ability to remove or add information to the state. On the other hand, LSTMs may lead to better results when the dataset is large. Sometimes bidirectional LSTM or bidirectional GRU is used to preserve information from both past and future, where the two directional neurons do not have any interactions.

2.4 One-hot Encoding Method

Our proposed models take raw DNA sequences (or RNA sequences) as inputs, which are encoded into bit matrices using one-hot encoding method with each nucleotide represented as a four-element binary vector. We encode the entire DNA nucleotides sequentially, where each nucleotide is encoded to a four-element binary vector with only one element set to one and others set to zero: A=(1,0,0,0), C=(0,1,0,0), G=(0,0,1,0), and T=(0,0,0,1). Then a DNA sequence will be represented as an encoded bit matrix, with columns corresponding to A, C, G, and T. With the one-hot encoding method, we can preserve the vital position information of each nucleotide in DNA sequences.

2.5 Deep Learning Platform for Experiments

We have implemented all the deep learning models with Python, Keras (Version 2.0.8) [32], and Theano (Version 0.9.0) [33]. Keras is a powerful python library which

can run on top of Theano [33] or TensorFlow [34]. It provides highly modularized APIs for building and training deep learning models. Theano is a linear algebra compiler which optimizes mathematical computations and provides efficient low-level implementations. It was developed at the University of Montreal for research and development into state-of-the-art deep learning algorithms. It handles operations on multidimensional arrays and has several optimizations including the use of GPU of computations. We configure the experiment environment and run the python codes on a Linux server with 4 Intel E5-2650 CPUs, 256 GB memory and 4 Nvidia GeForce GTX 1080 GPUs (2560 NVIDIA CUDA cores, 8 GB GDDR5X memory and 10Gbps memory speed).

2.6 Model Training and Testing

We divide the whole procedure of model training and testing into three steps: Firstly, we configure and apply different hyperparameters to train the models. Secondly, during the training process, we aim to discover the parameters (also called weights) that can minimize the objective function, which is challenging due to the high dimensionality and non-convex. We tune the hyperparameters to find out the weights with the best performance on the validation set for the model. Finally, the model is evaluated against other machine learning models and popular deep learning models on the test set. The training performance, as well as the training speed, significantly relies on parameter initialization, learning rate, and batch size of stochastic gradient descent. The optimum batch size has a relationship with learning rate, for instance, larger batch sizes commonly requiring smaller learning rates. We also employ a low learning rate decay in model training, which helps explore confined parameter regions of the objective function and avoids overshooting. In our proposed model, model weights are initialized and sampled from a truncated normal distribution centered on zero with the square root of the average number of both the input units and the

output units of the input layer [35, 36]. Since the model aims to classify the input into two categories: sequences with poly(A) sites and sequences without poly(A) sites, it is a two-class logistic regression problem. Thus, we prefer to adopt a sigmoid function rather than a softmax function that is used for a multiclass problem. Advanced adaptive learning rate methods, such as RMSprop, Adagrad [37], and Adam [38], are also applied to reduce the effect of initial and potentially sub-optimal learning rate for model training. Based on the preliminary experiments, the hyperparameters, such as a standard stochastic gradient descent optimizer, a learning rate of 0.001, a batch size of 128 and a momentum rate of 0.9, are appropriate as the start hyperparameters for training our models. Regularization, ensemble learning and cross-validated evaluation are always used for reducing overfitting. To reduce overfitting, we adopt the most common regularization technique – dropout with a dropout rate of 40%. Another popular method we employed is "early stopping", which means the training process will be stopped automatically at the best point. Thus, the latest parameters that perform best on the validation set are chosen as soon as the training process stopped. Hyperopt [39] can automatically explore the hyperparameter space using Bayesian optimization. We applied Hyperopt to tune the hyperparameters for our models.

2.7 Prediction Assessment Metrics

Metrics, including sensitivity or recall (SN), specificity (SP), overall accuracy (ACC), Matthew's correlation coefficient (MCC), area under the receiver operating characteristic curve (AUC) and F-measure (F1), are adopted to evaluate competing methods. Sensitivity (SN) (true positive rate) measures the fraction of the true positive samples that are correctly predicted. Specificity (SP) (true negative rate) measures the fraction of the predicted tRNA samples that are correct amongst those predicted. Accuracy (ACC) measures an average performance on positive and negative samples. However, this measure does not consider the possible difference

between positive and negative samples. MCC considers the relation between correctly predictive positives and negatives as well as false positives and negatives. MCC is especially useful when the two classes are of very different sizes and which could be seen as a correlation coefficient between training and testing datasets. AUC is commonly used as a baseline to determine if the model is useful. F-score can be interpreted as the harmonic mean of precision and recall. The higher f-score is, the higher both precision and recall are. We adopt AUC to assess the evaluation and performance and usability of different deep neural networks models. Let TP, TN, FP, and FN denote true positive, true negative, false positive and false negative, respectively. The definitions of them are shown as follows, where true positive (TP) and true negative (TN) are equal to the number of correctly predicted tRNA sequences and pseudo-tRNA sequences respectively, false positive (FP) and false negative (FN) are the numbers of incorrectly predicted tRNA sequences and pseudo-tRNA sequences respectively.

$$SN = \frac{TP}{TP + FN} \tag{2.1}$$

$$SP = \frac{TN}{TN + FP} \tag{2.2}$$

$$AUC = \frac{1}{2}(SN + SP) \tag{2.3}$$

$$F1 = \frac{2TP}{2TP + FP + FN} \tag{2.4}$$

$$Accuracy = \frac{TP + TN}{TN + FP + TP + FN} \tag{2.5}$$

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TN + FN)(TN + FP)(TP + FN)}} \tag{2.6}$$

CHAPTER 3

COMPARATIVE STUDY OF DIFFERENT DEEP LEARNING APPROACHES FOR GENOMIC SEQUENCE PATTERN RECOGNITION

3.1 Introduction

There have been several attempts at generating particular sequence patterns (such as regular expression patterns) from text sequences. Early research into this task used rule-based techniques to create a natural language interface to regular expression writing [40]. A novel transformation-based algorithm for learning such complex regular expressions was proposed in [41]. Fernau described algorithms that directly infer regular expressions from positive data and characterized the regular language classes that can be learned this way [42]. Later he described algorithms that directly infer elementary forms of 1-unambiguous regular expressions from positive data [43]. However, current research used predefined features to represent sequences to learn a semantic parsing translation model, which requires in-depth domain knowledge and involves extensive manual feature engineering. In addition, the loss of each character's essential position information in the sequence would affect the performance of prediction. With the growing availability of large-scale datasets and advanced computational techniques, a lot of research works apply deep learning models to understand genome regulatory instructions directly from gene sequences without pre-defined features, to predict the unknown sequences profile [22, 24, 25]. In view of this, we consider utilizing deep learning models for a more general problem: sequence pattern recognition. We intend to use the sequences themselves as inputs for training. We generated several datasets of sequences with different patterns and trained the two constructed deep learning models based on the simulated datasets.

We consider two major types of deep learning models - CNN and RNN for our genomic sequence pattern recognition problem. It is more challenging to train RNN than CNN because that an incorrect parameter initialization can lead to vanishing or exploding gradients. The models we applied do not separate the feature extraction and model training. They learn predictive sequence patterns in a data-driven manner. In this work we make three contributions: firstly, we apply deep learning to recognize different kinds of sequence patterns, which is a novel approach for sequence pattern recognition. Secondly, it is shown that deep learning models could automatically learn patterns without involving extensive manual feature engineering, which saves a lot of efforts. Through the experiments on simulated data, we find out that both CNN and RNN models can get outstanding results. Besides that, CNN is slightly better than RNN for our sequence pattern classification. Furthermore, we apply a saliency map to visualize the convolutional kernels of the first convolutional layer, in order to show the learned sequence patterns. The third contribution is shown that even with real genomic sequence data, deep learning methods can also get good results. Similarly, the CNN model performs a little better than the RNN model. The visualization by saliency map shows that our models can learn the strong signals in genes, such as start codon and stop codon. Note that the codes of the proposed methods are freely available on Github [1].

3.2 Model snd Pattern Design

3.2.1 Comparison Models

To offer fair comparisons to competitive models, we conducted a series of experiments with both traditional machine learning methods and deep learning methods. We tried our best to choose models that can provide comparable and competitive results, and the results are reported faithfully without model selection.

[1] https://github.com/stella-gao/DeepPat

We refer to traditional methods as those that using a hand-crafted feature extraction method and a linear classification method. The classification used is a binary classification for all these models. The involved state-of-the-art traditional machine learning methods include Support Vector Machine (SVM), Logistic Regression (LR), Decision Tree (DT) and Random Forest (RF). We apply these methods to the Kmer features extracted from the raw sequence datasets. We extract Kmer (K = 4) features from raw sequences, where Kmer means K sequence bases together. Thus, we need to find out all the frequencies of 1mer, 2mer, 3mer and 4mer as features and consider those as the input to our machine learning models. Here, 1mer features are the frequencies of A, C, G, and T; 2mer features are the frequencies of AA, AC, AG, AT, CA ... TT; 3mer features are the frequencies of AAA, AAC, AAG, AAT, CAA ... TTG, TTT; 4mer features are the frequencies of AAAA, AAAC, AAAG, AAAT, CAAA ... TTTG, TTTT. There are 4+16+64+256=340 features in total. Thus, each feature can be thought as a frequency of each kmer component.

Recent deep learning methods have started to be applied to text classification. We choose two simple and representative models for comparison, in which one is the convolutional neural network (CNN), and the other is the long-short term memory (LSTM) recurrent neural network (RNN) model. We also illustrate two deep learning models for our sequence pattern recognition problem: a CNN model, which consists of two convolutional layers and one max-pooling layer to identify predictive characters from the context of sequences and one fully connected hidden layer with 128 neurons to model their interactions; an RNN model, which consists of one LSTM layer which contains 128 neurons. We apply a task-specific activation function "sigmoid" to produce the probability predictions of sequence patterns existence, which is to be compared via a loss function Stochastic gradient descent (SGD) to the true value vector. For the RNN model, after hyperparameters tuning, we decide to apply "rmsprop" loss function to produce the probabilities, which is proven to be the best

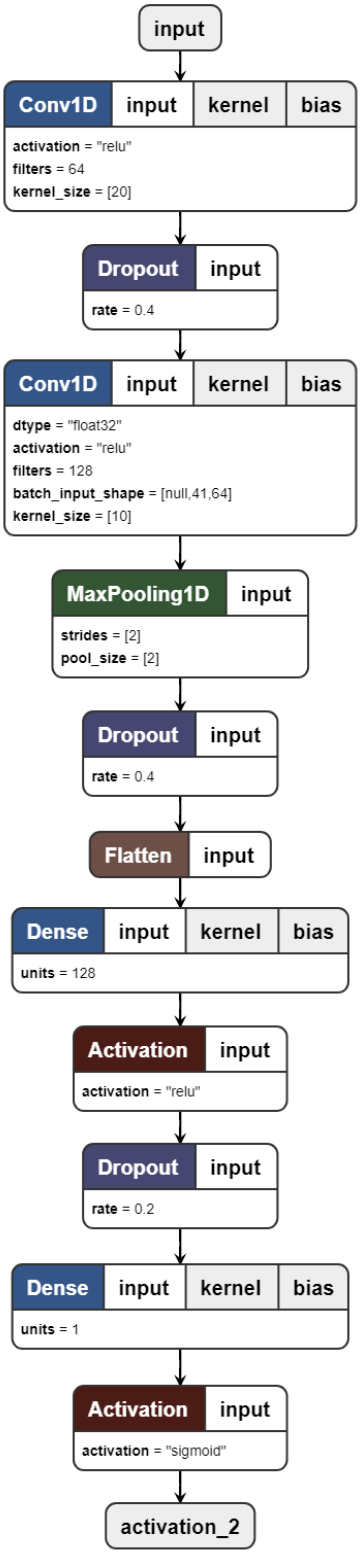


Figure 3.1 A graphical illustration of the CNN model on single-dimensional sequence data.

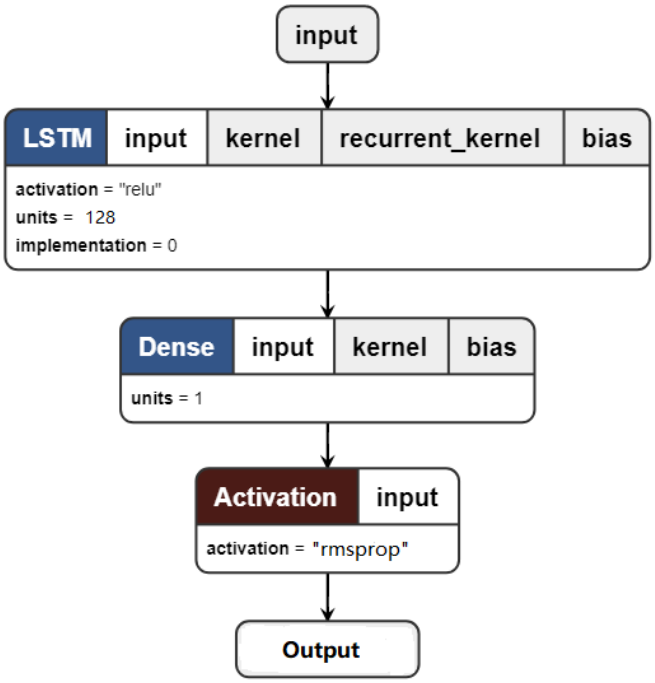


Figure 3.2 A graphical illustration of the RNN model on single-dimensional sequence data.

activation function for RNN models rather than SGD. Figure 3.1 and Figure 3.2 shows the graphical illustration of these models.

We illustrate the architecture of our CNN models for both the simulation experiment and real data experiment in both Tables 3.1 and 3.2. Both models consist of two convolutional layers and one max-pooling layer to identify sequence pattern or predictive motifs from raw sequence data and one fully connected hidden layers with a ReLU activation function to model motif interactions [27]. Techniques, like dropout [44], batch normalization [45] and early stopping, are employed to prevent overfitting. Table 3.1 shows the specification and hyperparameters of each layer in the simulation experiment, while Table 3.2 shows the specification and hyperparameters of each layer in the real data experiment. The hyperparameters (e.g., convolution kernel size, number of layers, dropout rate, etc.) are selected based on the performance on the validation dataset of each experiment. The raw genomic-format text sequence and

Table 3.1 CNN Model Architecture and Hyperparameters for the Simulated Sequence Dataset

Layer No.	**Layer Type**	**Size**	**Output**
0	INPUT	-	4*50
1	CONV	64*4*20	32*31
2	RELU	-	32*31
3	DROPOUT	-	32*31
4	CONV	128*4*10	64*22
5	RELU	-	64*22
6	POOL	4*2	64*11
7	DROPOUT	-	64*11
8	FC	128	128
9	DROPOUT	-	128
10	FC	1	1
11	SIGMOID	1	1

Table 3.2 CNN Model Architecture and Hyperparameters for the Genomic Sequence Dataset

Layer No.	**Layer Type**	**Size**	**Output**
0	INPUT	-	4*1000
1	CONV	32*4*64	32*937
2	RELU	-	32*937
3	DROPOUT	-	32*937
4	CONV	128*4*16	128*922
5	RELU	-	128*922
6	POOL	4*2	128*461
7	DROPOUT	-	128*461
8	FC	128	128
9	DROPOUT	-	64
10	FC	1	1
11	SIGMOID	1	1

gene sequence are both encoded into a bit matrices using one-hot encoding method. A 50-character genomics-format sequence can then be represented as an encoded 4×50 bit matrix, with columns corresponding to A, C, G and T. A 1000-nt DNA sequence can then be represented as an encoded 4×1000 bit matrix. The encoded bit matrix is passed to a one-dimensional convolutional layer, which acts as a motif scanner across the input matrix and computes the activation of multiple convolutional filters at every position within the sequence window. With the one-hot encoding method, we can preserve the vital position information of each nucleotide in DNA sequences.

3.2.2 Sequence Pattern Design

In this paper, we design six kinds of sequence patterns for mining sequence patterns. The designed patterns are shown in Table 3.3 and Figures 3.3. All the sequences are designed to have the same length, for example, 50 characters in our simulation experiment. We design a special pattern and randomly generate a number of sequences based on the special pattern. For each pattern, we generate positive samples with the specific sequence patterns and generate two kinds of negative samples: one kind is random sequences exclude the positive sequence pattern samples, the other kind is misleading sequences that are generated based on a similar pattern to the positive samples' pattern but has some different parts. The starting position can be randomly generated or fixed at a specific position in the sequence. If the starting position of the samples is randomly generated, that means the pattern in each sequence may locate at different positions. If the starting position is fixed, then all the patterns in the sequences start from a same position. For some patterns, the length of the patterns are also not fixed, which is based on the special pattern in the sequence. We design the patterns like that to make the pattern challenging enough for those models to learn. The sequence step length for this experiment is 50. For example, in pattern No. 1, the designed motif pattern [A, A, A, A, A],

allows up to 4 steps for each motif. And randomly insert these patterns into an individual sequence with no overlapping. If each motif pattern occurred only once in a sequence labeled as positive, otherwise labeled as negative. We generated five thousand positive samples and five thousand negative samples. The deep neural network must first recognize what is the motif and then learn to identify if the motif occurred once or not. This pattern is as identical as finding low-level feature then combined as a high-level feature.

We design the sequence pattern No. 5 as ACGTAAGGTC with each base randomly inserted into the sequence of length 50. We can easily design positive data. But the negative data of random sequences cannot just be random. Since the positions of each base of the sequence pattern are unknown, the randomly generated characteristic could be the same as our sequence pattern. It is most likely that a totally randomly generated sequence contains our sequence pattern. Thus, we design an algorithm to make sure the generated negative data cannot contain the sequence patterns. In the removing sequence pattern (RMP) algorithm, we need to generate N sequences of length L. Each time we generate a random sequence, we need to detect whether the sequence contains our pattern. We should search from the beginning of the sequence. When the sequence contains all the pattern base in order, we need to ignore this sequence and generate a new one until we obtain N sequences in total.

3.3 Experiments and Results

In this section, we describe the experimental settings, the platform, and packages used for building models as well as the evaluation performance comparing with the acknowledged state-of-the-art approaches. My experiments intended to illustrate different characteristics of the two different kinds of neural networks (convolutional neural network, recurrent neural network), and which network achieves a better performance than another under different scenarios.

Table 3.3 Description of the Designed Sequence Patterns

No.	**Pos/Neg Data**	**Length**	**Starting Position**	**Pattern Length**
1	pos	50	random	5 - 10
	misleading	50	random	1 - 4, 11 - 14
2	pos	50	fixed	10*2
	neg: mutate 50%	50	fixed	10*2
3	pos: ACAGT with fixed interval 9 bases	50	fixed	5
	neg: mutate one of the 5 pattern bases	50	fixed	5
4	pos: any combinations of A, C, G or T	50	fixed	4
	neg: mutate one of the 4 pattern bases	50	fixed	4
5	pos: ACGTAAGGTC with each base randomly inserted in to the sequence	50	random	10
	neg: mutate one of the 10 pattern bases	50	random	10
6	pos: randomly generate 10 bases and swap AT, swap GC for the remainder half	50	fixed	50
	neg: randomly select 50% bases from the second half to mutate	50	fixed	50

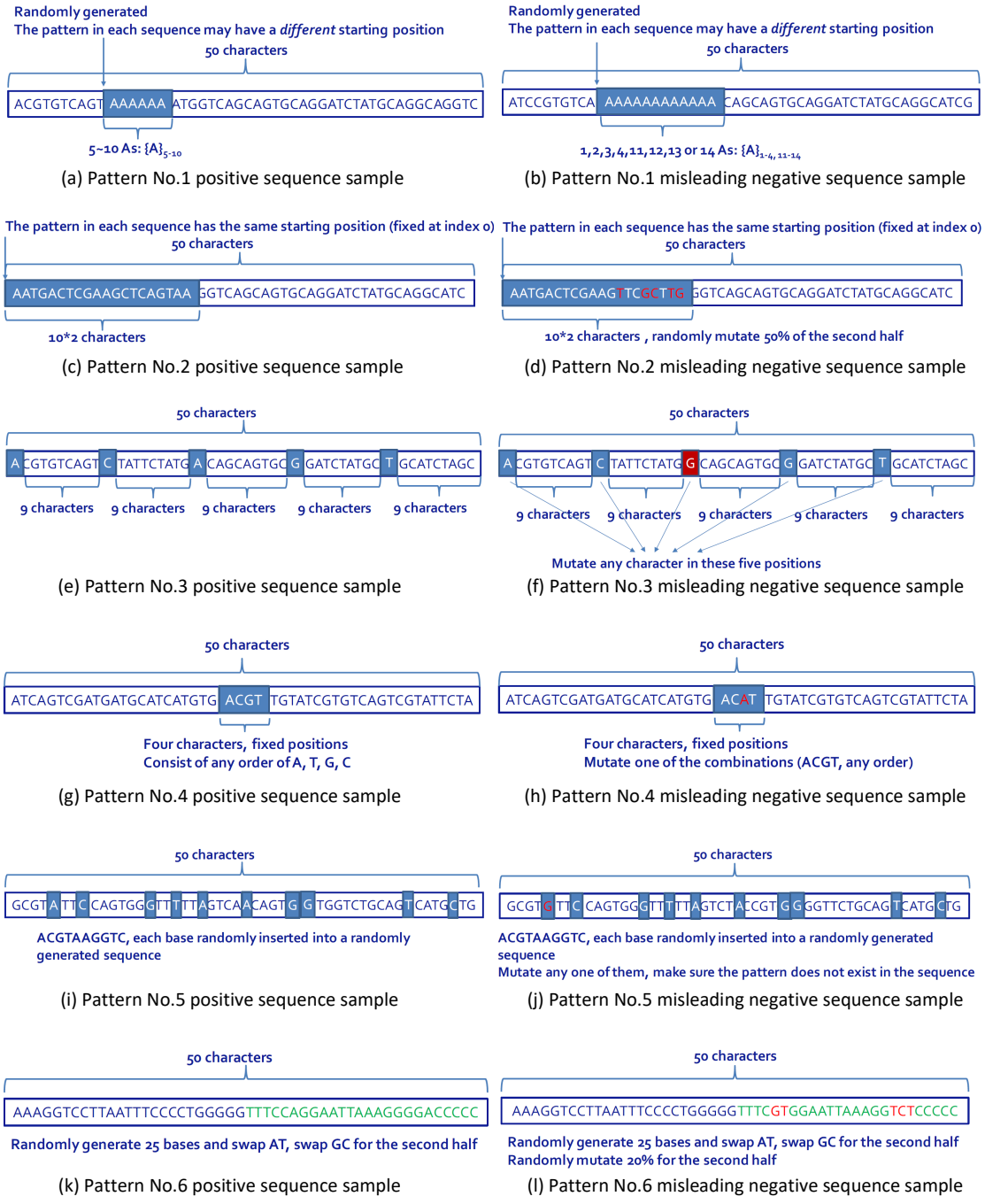


Figure 3.3 The positive and misleading negative sequence samples of the designed patterns.

3.3.1 Data Source and Data Preprocessing

All simulation experimental samples are sequences with a length of 50 base pairs and belong to "Positive" or "Negative" class. Samples in "Positive" class contain regions wrapping around sequence patterns. In contrast, samples in "Negative" class do not contain them. For simulated data, we design positive sequence pattern to randomly generate the positive samples and the misleading negative sequence patterns to randomly generate the negative samples. We randomly generate random negative sequence sample, while excluding the positive samples that might be inside those sequences. We apply a 10k size dataset to each experiment. Both the size of positive samples and the size of negative samples are the same. The model which performs best will then evaluated on the test set to quantify the performance. For each pattern, we randomly generate a different dataset based on a different seed that consists of both positive sequences and negative sequences. The seeds are 1009, 1013, 1019, 1021, and 1031. Thus, we repeat five times based on five different seeds for each sequence pattern.

For the real data experiment, we download the real caenorhabditis elegans gene gff3 file from ENSEMBL database [2]. Then we extract the fasta format sequence files using BEDTool [46] based on gff3 files. Since the dataset is large, we extract the gene sequences that are less than 1000 for our experiment. We conduct two experiments: one with the sequences whose lengths are less than 500nt and the other with the sequences whose lengths are less than 1000nt. As we observed, the sequences belong to seven chromosomes: chrI, chrII, chrIII, chrIV, chrV, chrX, chrMtDNA. As the number of gene sequences in chrMtDNA is too small (less than 20), which will not affect the experiment results. Thus, we just ignore these sequences. To make the training dataset, validation dataset and test dataset ratio similar to 7:1:2, we pick

[2] ftp://ftp.ensembl.org/pub/release-92/gff3/caenorhabditis_elegans/

the sequences of chrII, chrIV, chrV and chrX as training dataset; chrIII as validation dataset and chrI as test dataset for the experiments.

For this experiment, we split the entire dataset as training data, validation data and test data to avoid overfitting and assure that the model will generalize to new data. We holdout validation dataset and partition our entire dataset into a training set, a validation set, and a test set. Weights and parameters of the models are both learned from the training set and evaluated on the validation set.

3.3.2 CNN and RNN Model Training and Testing

After data preparing, we applied the designed deep learning models with different architectures in Section 2 on our prepared data. The goal of model training is to find the parameters that can minimize the objective function. In general, model parameters should be initialized randomly to avoid local optima determined by a fixed initialization [35, 36]. To train the models, we need to tune the hyperparameters (such as the number of hidden neurons, the number of kernels, the kernel size, the dropout rate, the learning rate, etc.) based on the performance of validation dataset. The idea of dropout is to randomly set output of the last layer to zero to prevent units from co-adapting too much during the training process and will disable during validation and testing process. The models will evaluate the average multi-task cross entropy loss on the validation set at the end of each epoch to monitor the progress of training. We set the parameters to schedule the training times of the models for 1000 epochs but may early stop if the validation loss did not decrease over 100 consecutive epochs. Dropout is the most common regularization technique and often one of the key ingredients to training deep models. The dropout rate corresponds to the probability that a neuron is dropped out, where 0.5 is a sensible default value. Between these two convolutional layers, we used dropout to randomly exclude 40% of neurons in the layer in order to reduce overfitting. Various learning rates are

usually explored on a logarithmic scale such as 0.1, 0.01, or 0.001, with 0.01 as initial training value [47]. Here, training is stopped as soon as the validation performance starts to saturate or deteriorate, and the parameters with the best performance on the validation set are chosen. We save the best model based on loss value of validation dataset with the patience of 100 epochs.

In our simulation experiment, the loss function of CNN was optimized by mini-batch stochastic gradient descent with a batch size of 128 and a global learning rate within [0.01, 0.1]. The learning rate was adopted by SGD and decayed by a default factor of 0.9 after each epoch. The loss function of RNN was optimized by RMSprop optimizer. As the number of configurations grows exponentially with the number of hyperparameters, trying all of them is impossible in practice. It is, therefore, recommended to optimize the most important hyperparameters such as the learning rate, batch size, or length of convolutional filters independently via line search (trying different values while keeping all other hyperparameters constant). Hyperopt [39] can automatically explore the hyperparameter space using Bayesian optimization, which we applied to tune the parameters for our models.

3.3.3 Experimental Environment and Assessment

We implement the proposed deep learning models with python, Keras (Version 2.0.8) [32], and Theano (Version 0.9.0) [33]. Keras [32] is a powerful python library which can run on top of Theano [33] or TensorFlow [34]. It provides highly modularized APIs for building and training deep learning models. We implement the traditional machine learning model with WEKA (Version 3.8). The Kmer features are extracted with python. We tuned the hyperparameters for our models using a tool named Hyperopt [39], which can automatically explore the hyperparameter space using Bayesian optimization. We configure the experiment environment and run the python codes on a Linux server with 4 Intel E5-2650 CPUs, 256 GB memory and 4 Nvidia

GeForce GTX 1080 GPUs (2560 NVIDIA CUDA cores, 8 GB GDDR5X memory and 10Gbps memory speed).

Six major performance metrics are used to evaluate the prediction method. They are the area under the receiver operating characteristic curve (AUC), recall or sensitivity (SN), Specificity (SP), Accuracy (ACC), Matthew's correlation coefficient (MCC) and F-measure. By applying the recognition function, we identify correctly TP (true positive) samples and TN (true positive) samples. At the same time, FP (false positive) positive samples were incorrectly classified as negative samples and FN (false negative) negative samples were incorrectly classified as positive samples. SN measures the fraction of the true positive samples which are correctly predicted. SP measures the fraction of the predicted positive samples which are correct amongst those predicted. Accuracy (ACC) measures the average of positive and negative samples, without considering the possible difference inside them. A better measurement MCC considers the relation between correctly predictive positives and negatives as well as false positives and negatives. MCC measures a correlation coefficient between trained and tested datasets. In addition, we adopt AUC to assess the performance as well. AUC is the expectation that a uniformly drawn random positive is ranked before a uniformly drawn random negative, which is commonly used as a baseline to determine if the model is useful. F-measure is applied as the weighted harmonic mean of precision and recall. The higher f-score is, the higher both precision and recall are. For each performance metric, we considered the sequences containing sequence patterns as the positive class and the sequences with no sequence patterns as the negative class. As a binary classification, positive sequences specificity corresponds to the negative sequences sensitivity (and conversely).

3.3.4 Experiment on Simulated Data and Results

First, we applied the CNN and RNN models to the raw datasets and then applied Support Vector Machine (SVM), Logistic Regression (LR), Decision Tree (DT) and Random Forest (RF) to the Kmer features extracted from the raw datasets. Note that during the training, we do calculate these performance measures for each individual sequence pattern, but only use the train data. Ten-fold cross validation was performed during hyperparameters tuning step using validation data. To achieve a fair comparison, we used the same datasets for training and compared the performance among the proposed CNN and RNN models. We then tuned the model hyperparameters and picked the best ones for calculating four metrics: Accuracy, AUC, recall and MCC on testing data. Lastly, to better understand the learning behavior of CNN and RNN models, we created learning curves for these two models. To create the learning curves, we used 128 instances as the initial sample size and increased the sample sizes by a batch size of k=128 until we reached the total number of annotated instances. At each sample size we performed cross validation 128 times and calculated the average accuracies. The summary of experimental results evaluated based on the prediction assessments is shown in Figure 3.4. As shown in Figure 3.4(a), CNN and RNN models can always get better performance than traditional machine learning models. It seems that RNN is slightly better than CNN in recognizing pattern No. 1. Although for the dataset with random negative sequences, SVM and LR models can get goood results, they cannot recognize the dataset with misleading negative sequences. DT and RF seem to be robust, but their performance are not good as CNN and RNN models, especially for MCC and SP metrics. For pattern No. 2, as shown in Figure 3.4(b), CNN and RNN models can always obtain high AUC values, no matter for the dataset with random negative sequences or misleading negative sequences. It seems that CNN is slightly better than RNN in recognizing this pattern, esperically for recognizing misleading sequences. RF can achieve a better

performance than other traditional machine learning methods, but the performance is still far worse than CNN and RNN models. For pattern No. 3, as shown in Figure 3.4(c), CNN and RNN models can always obtain high performances among all metrics for both datasets. Their performances are similar. SVM and LR can recognize the pattern, but the performance are worse than deep learning models, especially for the dataset with misleading sequences. Moreover, DT and RF cannot recognize this pattern. For pattern No. 4, as shown in Figure 3.4(d), CNN and RNN models have similar results. They can always obtain high performances among all metrics, no matter for the dataset with random negative sequences or misleading negative sequences. All traditional machine learning methods (SVM, LR, DT and RF) have similar results as well, but their performance are much worse than deep learning methods. For pattern No. 5, as shown in Figure 3.4(e), both CNN and RNN can always obtain high performances among all metrics for both datasets. RNN performs better than CNN in recognizing this pattern. SVM and LR have similar results with the performance better than DT and RF, and worse than CNN and RNN. DT has the worst results among all models. For pattern No. 6, as shown in Figure 3.4(f), both RNN and RF models can always obtain high performances among all metrics for both datasets. CNN can always obtain high performances among all metrics for the dataset with random negative sequences. But for the misleading dataset, CNN cannot recognize the pattern. RNN performs better than CNN in recognizing this pattern. SVM and LR cannot recognize the pattern for both datasets. RF performs better than DT in recognizing this pattern.

From the evaluation results, we can find that deep learning models perform better than traditional machine learning models for recognizing sequence patterns. Convolutional neural network exhibits high efficiency and high accuracy in identifying motif patterns. RNN models can be more appropriate for the sequence pattern where the character positions of the pattern are not fixed, since it may help find the

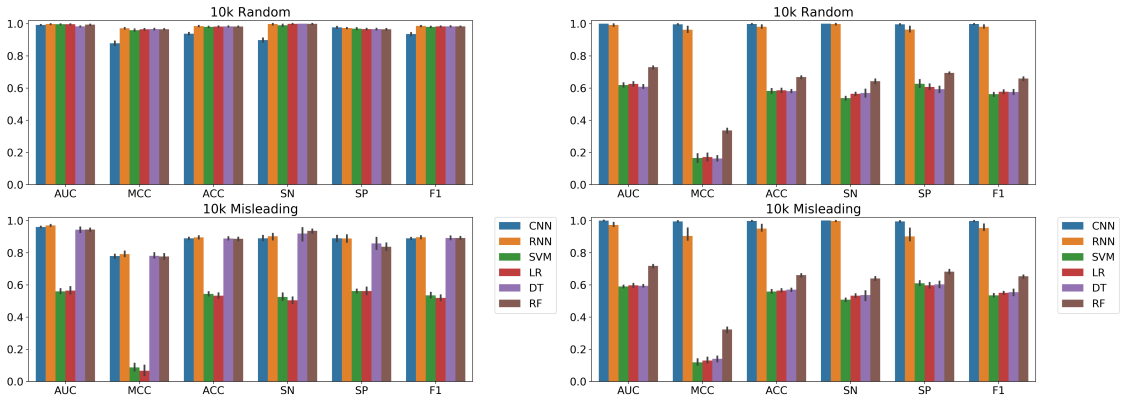
connection between the previous and next characters. Sometimes, RF can achieve a good performance, such as patterns No. 1, No. 2 and No. 6. Sometimes, SVM and LR can achieve a good performance, such as patterns No. 3 and No. 5. The traditional machine learning methods can be applied to some specific patterns, but it cannot achieve good performance for all sequence patterns. Thus, deep learning models are better than the traditional machine learning methods recognizing the sequence patterns. To make it more understandable, Figure 3.5illustrates the saliency map of the entire sequence with extracted sequence pattern based on CNN model, where a saliency map can only be visualized based on convolutional neural networks. It is observed that the signal is strong around the sequence pattern.

3.3.5 Experiment on Real Data and Results

The performance comparison between CNN model and RNN model is shown in Figure 3.6. The visualization based on saliency map method is shown in Figure 3.7. From the figures, we can discover that CNN model performs a little better than RNN model. Both models achieve a high AUC value and a high accuracy. The visualization illustrates how CNN learns the motifs from the raw gene sequences. Both the start codon (ATG) and the stop codon (TAG, TGA, TAA) has strong signals which can easily be learned from CNN model. In Figure 3.7, we can find the start codon at the positions 85, 88 and 91, and the stop codon at the positions 119, 128 and 131.

3.4 Conclusion

Genomic sequence pattern recognition problem is a kind of natural language processing problem. Since deep learning approaches do not need handcrafted features as input, it is always an efficient and effective way of solving this kind of problem with only raw sequences provided. To our knowledge, we are the first to compare different kinds of deep learning models for the genomic sequence pattern recognition problem.



(a) Prediction performance for learning pattern No.1

(b) Prediction performance for learning pattern No.2

(c) Prediction performance for learning pattern No.3

(d) Prediction performance for learning pattern No.4

(e) Prediction performance for learning pattern No.5

(f) Prediction performance for learning pattern No.6

Figure 3.4 Prediction performance for learning sequence patterns. Deep learning methods and state-of-the-art traditional machine learning methods are involved. Performance is measured using barplot by six evaluation metrics, including the area under the receiver-operating characteristic curve (AUC), accuracy (ACC), recall (SN), specificity (SP), Matthew's correlation coefficient (MCC), and F-score (F1).

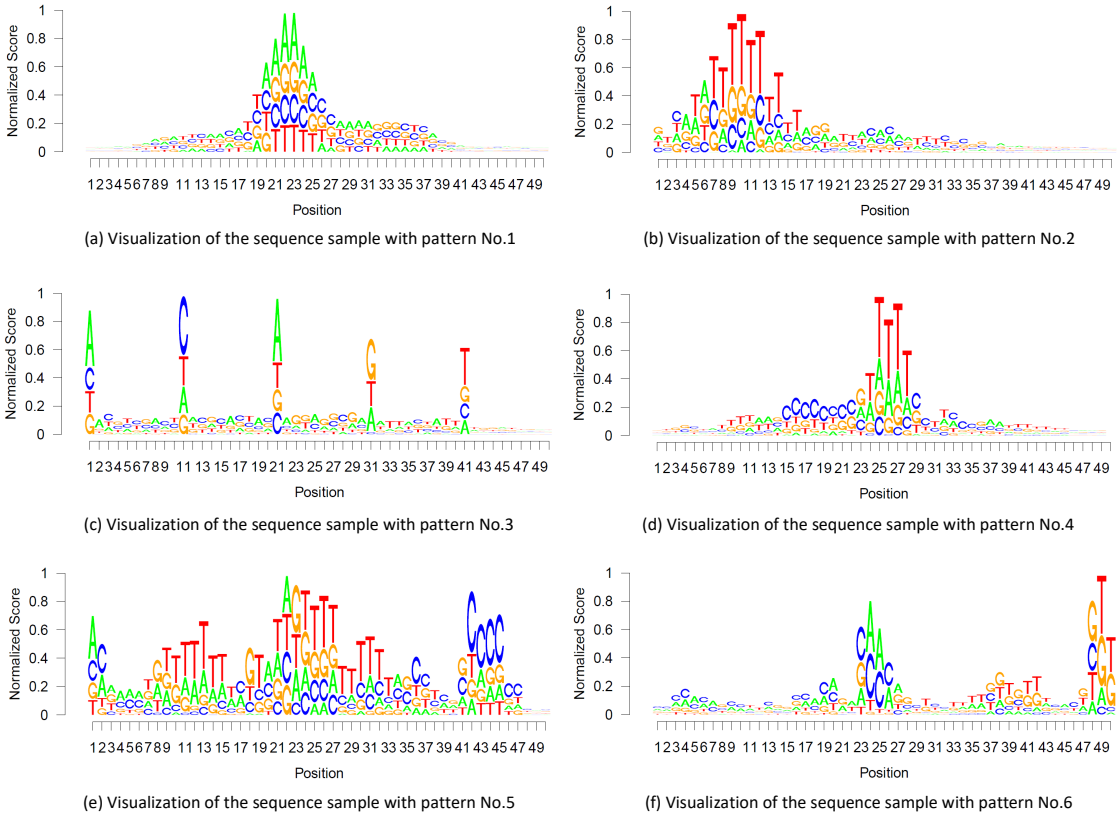


Figure 3.5 Visualization of the sequence samples with the patterns based on Saliency Map.

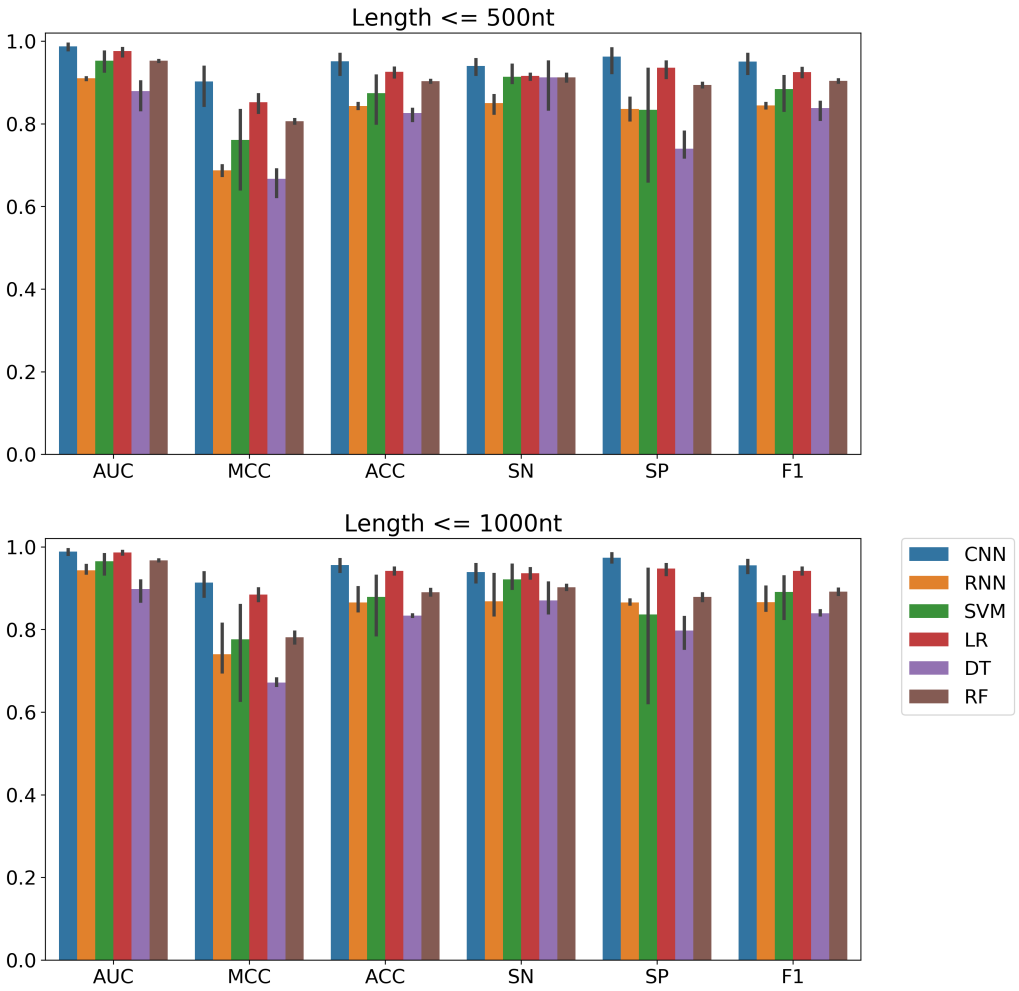


Figure 3.6 Prediction performance for learning gene sequence patterns. Deep learning models and state-of-the-art traditional machine learning methods are involved. Performance is measured using barplot by six evaluation metrics, including the area under the receiver-operating characteristic curve (AUC), accuracy (ACC), recall (SN), specificity (SP), Matthew's correlation coefficient (MCC), and F-score (F1).

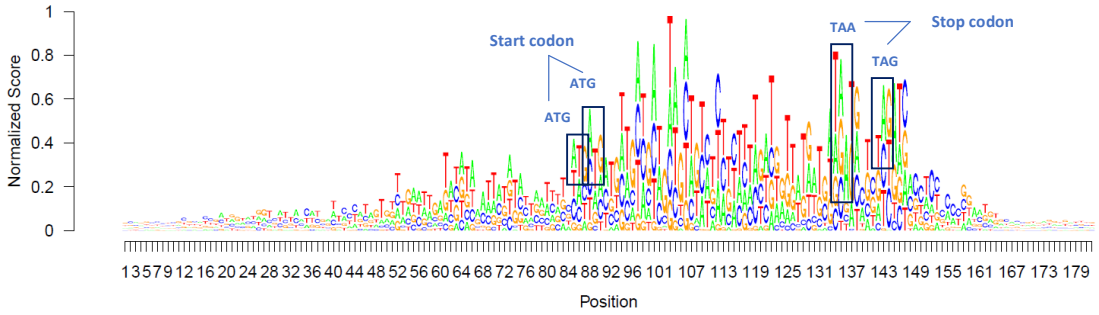


Figure 3.7 Visualization of real gene sequence by saliency map.

To make the problem simple, we generate all the sequence data in genomics format. The sequences are then transformed into bit matrices as the input to deep learning models using one-hot encoding method. We apply two dominant deep learning models and then compare them with different traditional machine learning methods on Kmer features for characterizing different forms of sequence patterns. Through the simulation results, we can find out that convolutional neural network can always get a good result on genomic format sequence pattern classification, except for some special sequence patterns that are difficult to discover. In view of this, we also apply deep learning models to real caenorhabditis elegans gene sequences and discovered that convolutional neural network achieved a better performance than the recurrent neural network. Through the visualization by saliency map, the convolutional neural network can learn the start codon and stop codon pattern of gene sequences.

CHAPTER 4

DEEPPOLYA: A CONVOLUTIONAL NEURAL NETWORK APPROACH FOR POLYADENYLATION SITE PREDICTION

4.1 Introduction

Polyadenylation is a vital process that occurs after gene transcription and produces mature messenger RNA (mRNA) for translation by synthesizing the polyadenylation tail at the RNA's 3'-end [14]. Recent discoveries have revealed that the 3'-end of most protein-coding and long-noncoding RNAs (lncRNAs; noncoding transcripts of 200 nucleotides or longer) is cleaved and polyadenylated [13]. In addition, alternative polyadenylation (APA) is prevalent in all eukaryotic species and plays critical roles in gene regulation, especially in the processes such as mRNA metabolism, protein diversification and protein localization [12]. Specifically, in addition to conducing to the intricacy of transcriptome by producing isoforms of distinct properties, it can regulate the translation efficiency, function, stability and localization of target RNAs [48, 13]. The polyadenylation site, also called the poly(A) site, is defined by surrounding RNA segments and conserved across metazoans with some minor variations in mammals [12, 13, 14].

Accurate prediction of poly(A) sites and identification of motifs that controlling them are fundamental for interpreting the patterns of gene expression, improving the accuracy of genome annotation and comprehending the mechanisms that governing gene regulation [15, 16]. Despite considerable advances in using machine learning techniques for this problem, its efficiency is still limited by the lack of experiences and domain knowledge to carefully design and generate useful features, especially for plants. With the increasing availability of extensive genomic datasets and leading computational techniques, deep learning methods, especially convolutional neural

networks, have been applied to automatically identify and understand gene regulation directly from gene sequences and predict unknown sequence profiles.

This chapter presents DeepPolyA, a new deep convolutional neural network-based approach, to predict polyadenylation sites from the plant Arabidopsis thaliana gene sequences. We investigate various deep neural network architectures and evaluate their performance against classical machine learning algorithms and several popular deep learning models. Experimental results demonstrate that DeepPolyA is substantially better than competing methods regarding various performance metrics. We further visualize the learned motifs of DeepPolyA to provide insights of our model and learned polyadenylation signals.

4.2 Motivation and Related Work

However, this remains a challenging problem, especially for plants, to precisely identify the poly(A) signals and predict poly(A) sites. Unlike animals, plants possess much less conserved signal sequences in such regions [49]. For example, the upstream element signal "AAUAAA" (or "AATAAA" in DNA sequence), which has been identified as the best signal in plants, can only be found in approximately 10% of Arabidopsis genes [50, 51]. In contrast, the same signal is utilized by 50% of human genes [52]. The variable structures composed of functional motifs [53, 54] also increase the difficulty in identifying poly(A) sites. In addition, because of the epidemic presence of alternative polyadenylation in intron and coding sequence (CDS), the poly(A) sites may locate in the genomic regions other than 3' untranslated region (3'-UTR). Thus, an ideal predictive model should be powerful and robust enough to overcome all barriers as mentioned above to achieve decent performance.

Quite a few methods have been proposed to predict poly(A) sites across diverse species. Among these studies, most of them focus on human sequences. Akhtar et al. proposed POLYAR, which applied the linear discriminant function (LDF) to

classify poly(A) sites into three groups with distinct poly(A) signals [55]. A stand-alone program named polya_svm was developed for poly(A) sites prediction using the 15 cis-regulatory elements based on a Support Vector Machine (SVM) model [56]. Chang et al. proposed a predictive model of two SVMs for features extraction and poly(A) sites prediction [57]. All these three studies are conducted based on human genomic sequences dataset polya_DB[1] [58]. More recently, Xie et al. proposed a novel machine-learning method by marrying generative learning (hidden Markov models) and discriminative learning (support vector machines) [59]. Methods for analyzing other species, such as yeast and plants, were also proposed. Graber et al. proposed a contextual model to predict yeast poly(A) sites via a hidden Markov model (HMM) [60]. In view of the features of plant poly(A) signals, Ji et al. proposed a generalized hidden Markov model (GHMM) to effectively predict the poly(A) sites in Arabidopsis genes [49]. It yielded both high specificity and sensitivity in the testing datasets [50]. Later, the model was updated and re-trained for rice [61]. Recently, Ji et al. proposed a user-friendly framework called poly(A) site classifier (PAC) for predicting poly(A) sites in Arabidopsis genes [62]. PAC demonstrates the best performance with high specificity and sensitivity in the real data experiments. In addition, sub-models, like feature generation, feature selection and classification in PAC, could be replaced and updated, making it adaptable to different datasets [62].

Although methods mentioned above could achieve decent performance in solving the specific problem, researchers are required to carefully design and generate useful features based on their experiences and domain knowledge. Feature generation and extraction methods, including K-gram pattern, Z-curve, and position-specific scoring matrix, are critical components for previous SVM-based or HMM-based methods [49, 62], and the power of the method could be significantly reduced due to an

[1]http://polya.umdnj.edu/PolyA_DB1/

inappropriate feature generation procedure. Thus, special efforts are needed to apply one method to another species.

With the increasing availability of extensive genomic datasets and leading computational techniques, deep-learning-based methods have been proposed to automatically identify and understand regulatory regions of the genome directly from DNA sequences, and predict the profile of unknown sequences based on learned knowledge [27]. Deep learning, in general, refers to methods that learn a hierarchical representation and detect complex patterns from feature-rich datasets through multiple layers of abstraction.

Amongst a set of deep neural networks, convolutional neural networks (CNN) are extensively employed in both academia and industry. It can achieve superb results in computer vision, video analysis and speech recognition for its efficient feature extraction capability [20, 63]. CNN has also been applied as the premier model in piles of genomic problems, for example, motif discovery [22], HLA class I-peptide binding prediction [64], and identifying functional effects of noncoding variants [24, 25]. Zeng et al. proposed a series of CNN architectures to identify DNA sequence binding with a large compendium of transcription factor datasets [65]. Basset, a powerful computational tool, was proposed to apply CNN to discover the functional activity of genomic sequences [23]. Zhou et al. applied CNN model to capture the motif signals from the sequences around the target residues [66]. DeepSEA is a recently developed algorithm that utilizes CNN for predicting chromatin effects of sequence alterations with single-nucleotide sensitivity [24]. DanQ, a hybrid framework that combines convolutional and recurrent neural networks, further improves the performance of DeepSEA [25]. CNN models have demonstrated their advantages in automatically learning hierarchical feature representations of raw input data in previous studies. Their successes motivate us to develop novel CNN-based methods to automatically learn poly(A)-related features, signals, and patterns for predicting poly(A) sites.

Thus, we propose a computational method, DeepPolyA, based on deep CNN for predicting poly(A) sites in Arabidopsis species. DeepPolyA automatically combines the feature extraction and model training stages, and learns predictive DNA patterns and motifs in a data-driven manner. In this work, we have made four contributions:

1. We propose a CNN-based model named DeepPolyA [2] to predict poly(A) sites in Arabidopsis. To the best of our knowledge, this is the first deep learning based approach to this research issue.

2. We show in this chapter that DeepPolyA could automatically learn poly(A)-related motifs without involving any manual feature engineering. The model first learns low-layer features from DNA sequences via lower convolutional layers, and then forms high-level, sophisticated features through upper nonlinear transformation layers.

3. DeepPolyA outperforms not only the conventional machine learning methods including Support Vector Machine (SVM), Bayesian Networks and Random Forest, but also the existing deep learning models including DanQ [25], DeepSEA [24], and VGG [67] models.

4. We also investigate the performance of alternative deep learning architectures in predicting poly(A) signals, including the recurrent neural network (RNN) model and the combination of CNN and RNN models (CNN-RNN).

4.3 Architecture of DeepPolyA Model

For the poly(A) site prediction problem, we illustrate the architecture of our DeepPolyA model in Figure 4.1. It consists of two convolutional layers and one max-pooling layer to identify predictive motifs from the context of DNA sequences

[2]DeepPolyA is freely available on Github: https://github.com/stella-gao/DeepPolyA

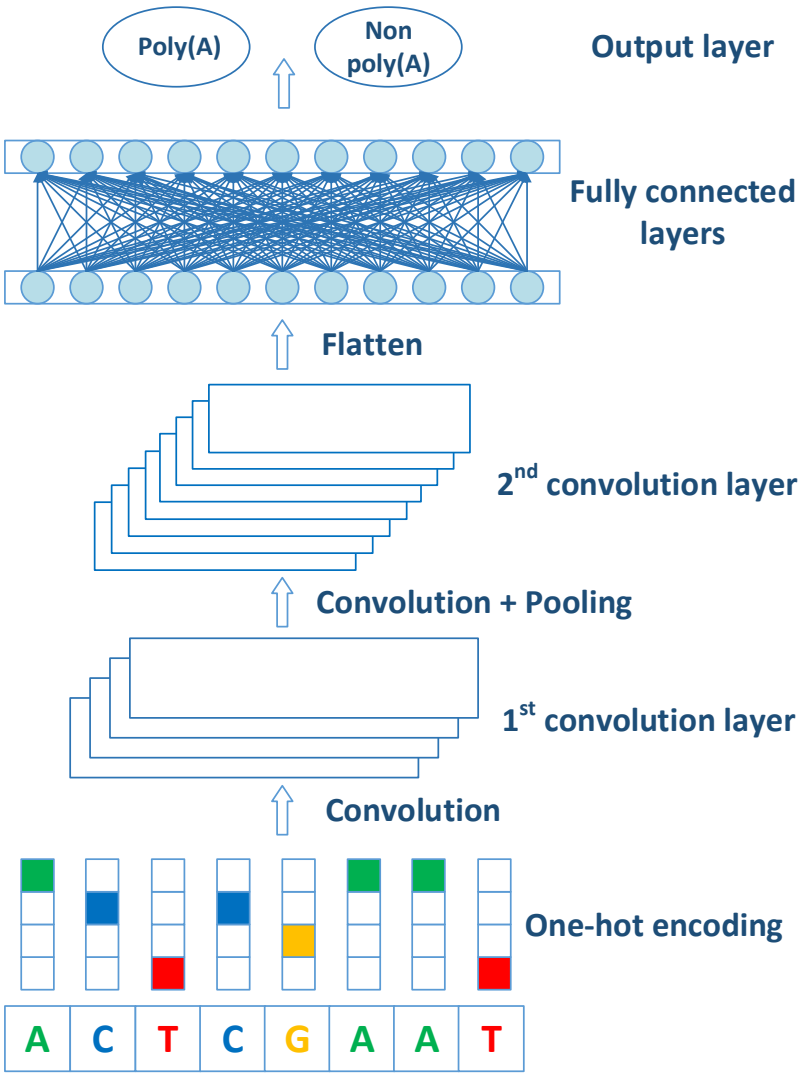


Figure 4.1 Overview of the neural network architecture of DeepPolyA.

and one fully connected hidden layers with a ReLU activation function to model motif interactions [27]. Techniques, like dropout [44], batch normalization [45] and early stopping, are employed to prevent overfitting. Table 4.1 shows the specification of each layer. The hyperparameters (e.g., convolution kernel size, number of layers, dropout rate, etc.) are selected based on the performance on a validation dataset. As shown in Figure 4.1 and Table 4.1, DeepPolyA takes a 162-nucleotide (nt) length DNA sequence as input with the position 131 as the target poly(A) site. The raw DNA sequence is encoded into a bit matrix using one-hot encoding method. A 162-nt

DNA sequence can then be represented as an encoded 4×162 bit matrix, with columns corresponding to A, C, G and T.

Table 4.1 DeepPolyA Architecture and Hyperparameters

Layer No.	Layer Type	Size	Output
0	INPUT	-	4*162
1	CONV	16*4*8	16*157
2	RELU	-	16*157
3	DROPOUT	-	16*157
4	CONV	64*4*6	64*152
5	RELU	-	64*152
6	POOL	4*2	64*76
7	DROPOUT	-	64*76
8	FC	64	64
9	DROPOUT	-	64
10	FC	1	1
11	SIGMOID	1	1

As shown in Table 4.1, the size column describes the kernel size of the convolutional layer, the window size of the max-pooling layer and the size of the fully connected layer. Note that the architecture parameters (e.g., the kernel size, the number of layers and the dropout rate) are carefully selected based on the optimization performance on the validation set. By applying several convolutional and pooling operations, CNN could automatically extract high-level features from high-dimensional input data while making the number of model weights manageable. Model parameters are randomly initialized as suggested by Glorot et al. [36]. Model hyperparameters (e.g., learning rate and the number of epochs) are optimized based

on the performance of the validation data. Note that validation loss is measured after each training epoch to monitor convergence. Dropout (dropout rate 40%; i.e., randomly drop 40% neurons in each iteration) and batch normalization [45] are used for additional regularization. Note that dropout is not suitable to be put in the last fully-connected layer, because some significant features may be lost. Thus, we only apply dropout between the hidden layers.

The first convolutional layer operates directly on the encoded bit matrix, where the kernels scan for features across the matrix, compute the activation of multiple kernels at every position within the DNA sequence window and generate output matrices. To reduce dimensionality and accelerate convergence, max-pooling layer and batch normalization layer are applied subsequently, where the pooling size and stride size are both set to 2 to prevent any overlap. A second convolutional layer is added to model the interactions between motifs generated by previous layers and obtain high-level features and abstractions. The output of the convolutional layers is then flattened into vectors and fed to the fully connected layer. Finally, the outputs are converted into probabilities via "sigmoid" function.

The value of the proposed deep learning based method is two-fold. Firstly, classical machine learning methods require researchers' prior knowledge and experience to predefine and cultivate features by counting or summarizing known genomic patterns (e.g., regulatory variants, k-mer and structural elements). In contrast, our method can automatically learn the problem-related knowledge and extract high-level features from the raw sequence data. Therefore, our approach could be readily applied to solve the same poly(A) prediction problem in different species. Secondly, the proposed method can capture nonlinear dependencies and interactions among the detected patterns and features at multiple genomic scales. It is challenging for even experienced researchers to design a schema of features including all potential

interactions of different sub-signals and low-level features. Thus, it can yield better performance as demonstrated in the real data experiments.

4.4 Experiments and Results

In this section, we describe the experimental environment, platform settings used for building models as well as the evaluation performance of the proposed model compared with the state-of-the-art approaches. The experiments show that DeepPolyA yields significantly more accurate predictions than existing baseline machine learning methods and other popular deep learning methods.

4.4.1 Data Source and Data Preprocessing

A large number of training samples are usually required to train deep neural networks in order to learn informative features and high-level representations from scratch. As a general guideline, the number of training samples should be at least as many as that of model parameters, although overfitting can be reduced through special architectures and model regularization techniques [47]. The same Arabidopsis datasets chosen from the baseline PAC's literature [50] were utilized for our experiments. Positive samples are randomly sampled from 16K dataset, which has over 16,000 Arabidopsis 3'-UTRs plus downstream sequences[3]. Following Ji et al [62], we selected the sequences, whose lengths are longer than 162 nt, for further analysis. As a result, 13427 positive sequences, which are equally distributed among five chromosomes (chr 1-5), are obtained. Negative samples are generated by randomly sampling sequences from the Arabidopsis Information Resources (TAIR) database[4], which consists of unequal-length introns, coding sequences and 5'-UTR sequences. The extracted negative samples contain 3222 introns, 9704 coding sequences and 501 5'-UTRs

[3]http://www.users.miamioh.edu/liq/links.html

[4]http://www.arabidopsis.org/

of Arabidopsis DNA sequences. Note that it maintains the same distribution of sequences as in the TAIR dataset and contributes 13427 negative sequences in total. As presented in [62], each sequence is then trimmed into a sequence of size 162 nt, containing 131 nt upstream and 31 nt downstream of a poly(A) site. After preprocessing, all experimental samples are 162nt-long genomic sequences. As shown in Equations 2.1- 2.6, we consider the sequences containing poly(A) sites as the positive samples and the sequences without poly(A) sites as the negative samples, and compute all the metrics accordingly.

4.4.2 Experiments on Model Comparison

We compare the proposed method with state-of-the-art methods including Support Vector Machine (PAC.SVM), Bayesian Networks (PAC.BN), Random Forest (PAC.RF) as well as other deep neural network models such as recurrent neural network (RNN), hybrid convolutional and recurrent neural network (CNN-RNN). Note that the architecture and parameters for RNN and CNN-RNN are tuned via validation set, which is exactly the same as for our proposed method. Figure 4.2 shows a graphical illustration of the RNN model, including two Long Short-Term Memory (LSTM) layers and one fully connected layer. Both LSTM layers contain 16 neurons and the fully connected layer contains 64 neurons. The dropout rate of each LSTM is set to 0.2. Figure 4.3 shows a graphical illustration of the CNN-RNN model, which is built with two convolutional layers, one max-pooling layer, one LSTM layer followed by one fully connected layer. To build a CNN-RNN model, we add one LSTM layer to DeepPolyA model between its second convolutional layer and the final fully connected layer. We also consider some other popular deep learning models (e.g., DeepSEA [24], DanQ [25], VGG [67]), fit them into our poly(A) problem, and compare them with our newly proposed CNN architecture (DeepPolyA).

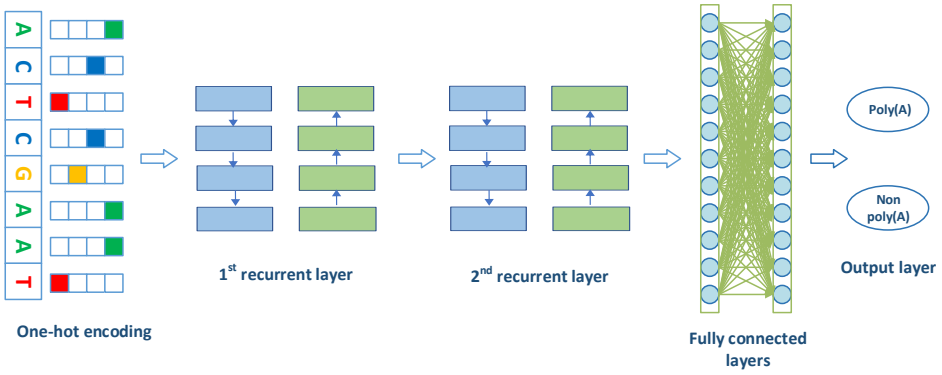


Figure 4.2 RNN model architecture.

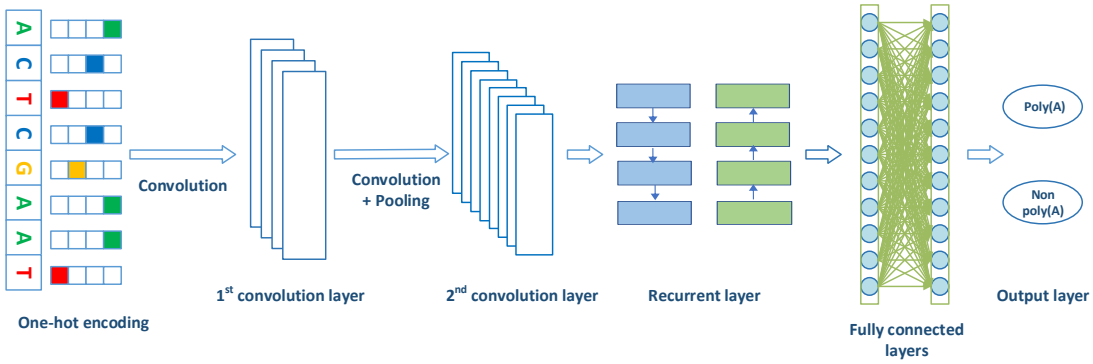


Figure 4.3 CNN-RNN model architecture.

For the traditional machine learning approaches (PAC.SVM, PSC.NB and PAC.RF), we follow Ji et al. [62] and extract features from DNA sequences, including 1) the frequency of some nearby nucleotides; 2) the hexamer weight of NUE region; 3) the PSSM-based CIS score of NUE region or CS region; 4) the components of Z-curve [62]. After feature extraction, we train the model based on different algorithms with the default parameter settings in Weka [68]. It should be noted that, as shown by the preliminary experiments, reasonable choices of the parameters (e.g. different type of kernels for Support Vector Machine, different number of decision trees for Random Forest) yield similar prediction results.

We divide the entire dataset into training, validation and testing sets. The training set is utilized to learn weights and parameters of the models, which are then evaluated on the validation set. The best model is selected and tested on the testing set to evaluate the model performance. Note that the data are partitioned on chromosome level and there is no overlap between training and testing data (i.e., three chromosomes for training, one for validation and one for testing). In each run, we randomly pick one chromosome as validation set and one chromosome as testing set, and leave the rest three chromosomes as the training set. We repeat the experiments several times to demonstrate that our model could achieve better performance, and it is not obtained by chance. All the hyperparameters of above-mentioned competing methods were selected based on the performance on the validation data. The model performance is then evaluated on the testing set.

As shown in Figure 4.4, we repeat the experiment five times and show the prediction performance based on testing data. DeepPolyA achieves remarkable and stable performance in classification on the testing set with an accuracy of 91.28%, a recall of 93.01%, a specificity of 89.51%, a Matthews correlation coefficient of 82.60%, an area under a receiver operating characteristic (ROC) curve of 97.06% and an

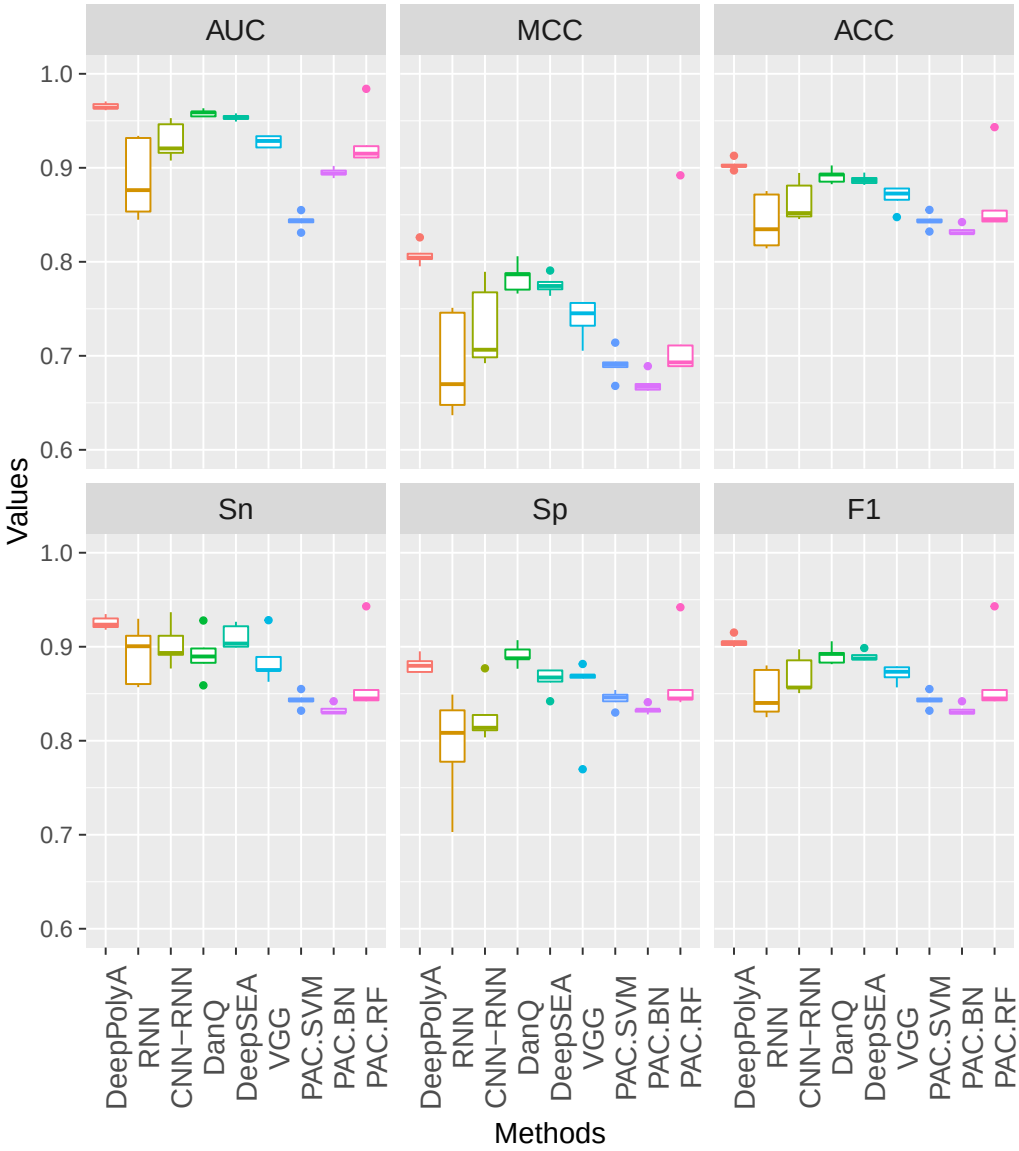


Figure 4.4 Prediction performance of DeepPolyA comparing with other methods. Performance is measured using boxplot by six evaluation metrics, including the area under the receiver-operating characteristic curve (AUC), accuracy (ACC), recall (Sn), specificity (Sp), Matthew's correlation coefficient (MCC), and F-score (F1). Three constructed deep neural network methods (including DeepPolyA), three referred popular deep learning methods and three baseline methods are considered. The lower and upper hinges are the 25 and 75 quartiles, respectively.

F-measure of 91.51%, which outperforms all other methods among all metrics. The two methods next to DeepPolyA are DanQ and DeepSEA.

The RNN method can get a high performance sometimes, but the result fluctuates according to different training and testing set. We can also discover that the performance of three baseline machine learning methods is always worse than deep learning methods. Overall, the results indicate that DeepPolyA can automatically learn high-level features from the DNA sequences and yields more accurate predictions than classical approaches and other popular deep learning models. A major advantage of our model compared to previous methods is its convolutional architecture, which allows for discovering predictive motifs in larger DNA sequence contexts, as well as for capturing complex patterns around poly(A) sites.

Figure 4.5 shows the area under receiver operating characteristics (ROC curve) and Figure 4.6 also shows the area under precision-recall curve (PR curve) for deep neural network based methods. We can find from the curves that DeepPolyA demonstrates the best performance. The following methods are DanQ, DeepSEA, CNN-RNN, VGG and RNN.

We further investigate the impacts of the length of input sequences by varying the input DNA sequence length from 54 nt to 216 nt. As shown in Figure 4.7, the performance of the proposed model increases when sequence length increases from 54 nt to 162 nt, and it keeps the same from 162 nt to 216 nt. Note that the performance will not increase if we further improve the input sequence length. These observations suggest that the sequences around poly(A) sites do contain the poly(A) signals and thus an appropriate input sequence length is critical for predicting the poly(A) sites (i.e., shorter sequence will miss some signals and longer sequences will include lots of noise).

We also evaluate the performance of our model in different genomic contexts. As shown in Figure 4.8, the model yields the best performance in CDS, and the worst

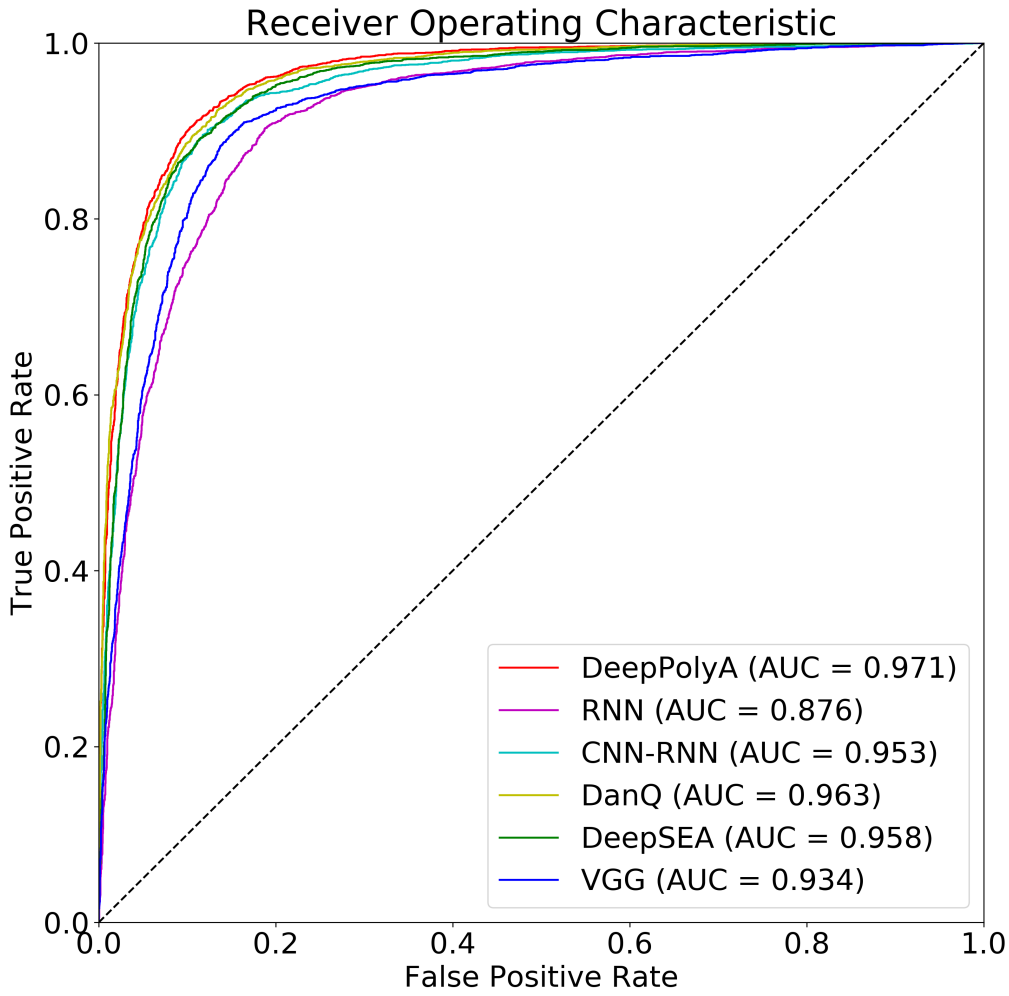


Figure 4.5 ROC curves of the prediction performance among all the deep learning methods.

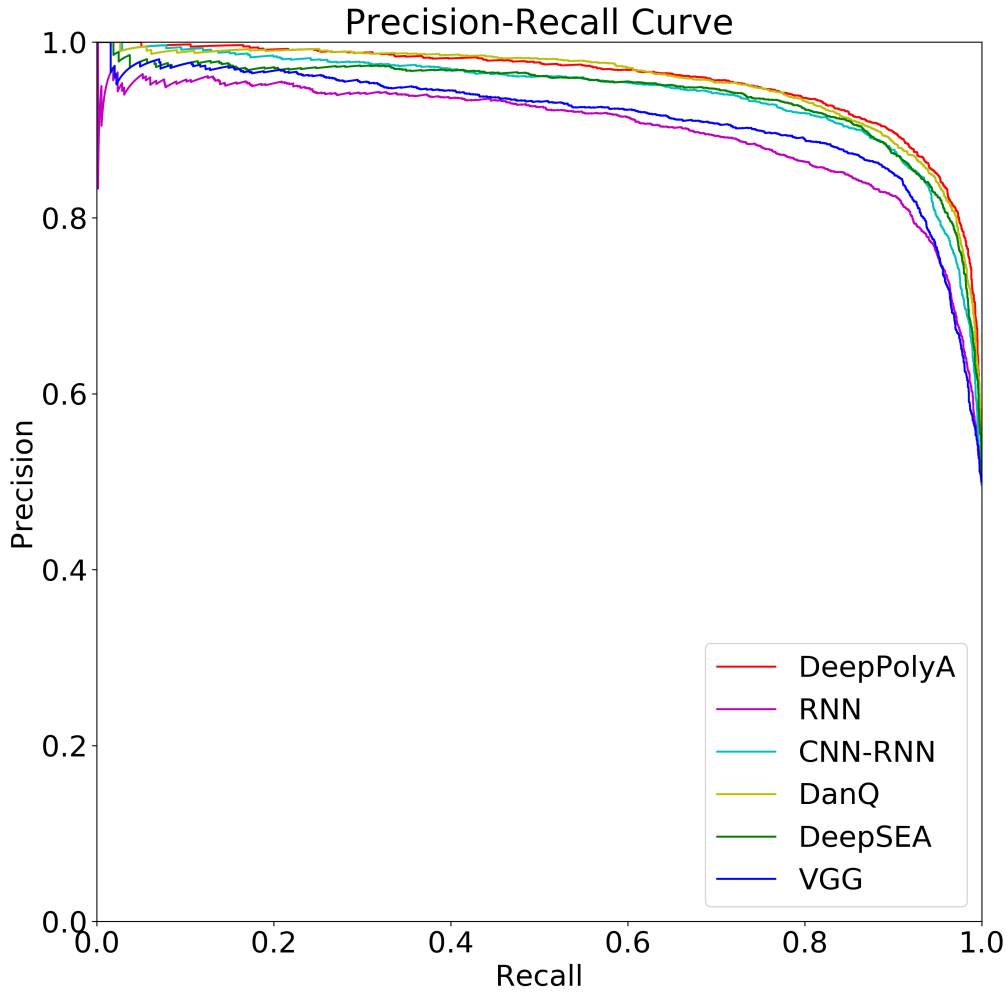


Figure 4.6 Precision-Recall curves of the prediction performance among all the deep learning methods.

performance in intron, suggesting that poly(A) sites located in introns may have less conserved signals and therefore are harder to detect.

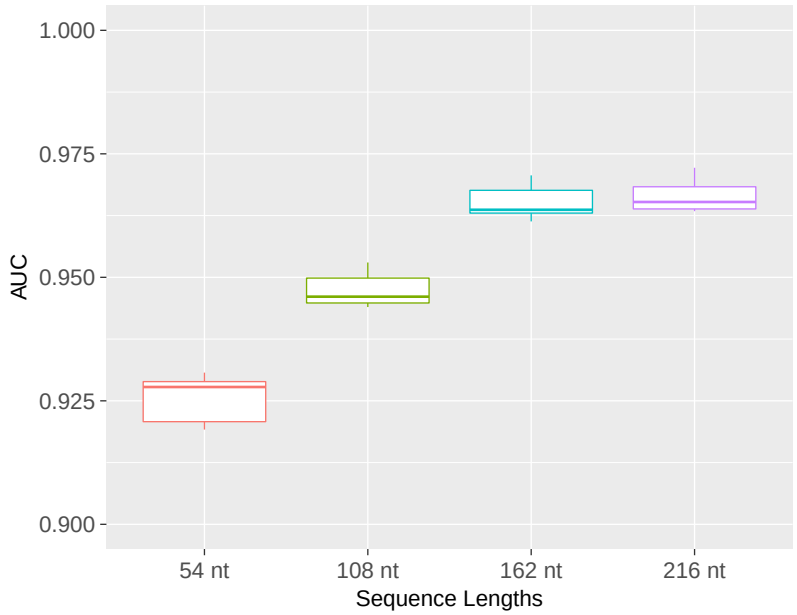


Figure 4.7 Prediction performance of DeepPolyA for DNA sequence windows of length 54 nt to 216 nt.

4.4.3 Visualization of Learned Motifs

Model visualization is critical in computational biology. Several approaches have been proposed to interpret the parameters of neural networks and to obtain insights into learned features. Similar to conventional position weight matrices (PWMs), gene sequence motifs can be recognized by the filters of the first convolutional layer in DeepPolyA and visualized as sequence logos. Traditional approaches for visualizing convolutional filters could be generally classified into alignment-based and optimization-based methods. Alignment-based approaches [22, 23, 25] align DNA sequence fragments that maximize the activation of a certain convolutional filter and visualize the outcome alignments as sequence logos using WebLogo [69]. Optimization-based approaches [70] optimize the input gene sequence to maximize the activation of a certain convolutional filter by gradient descent. Following

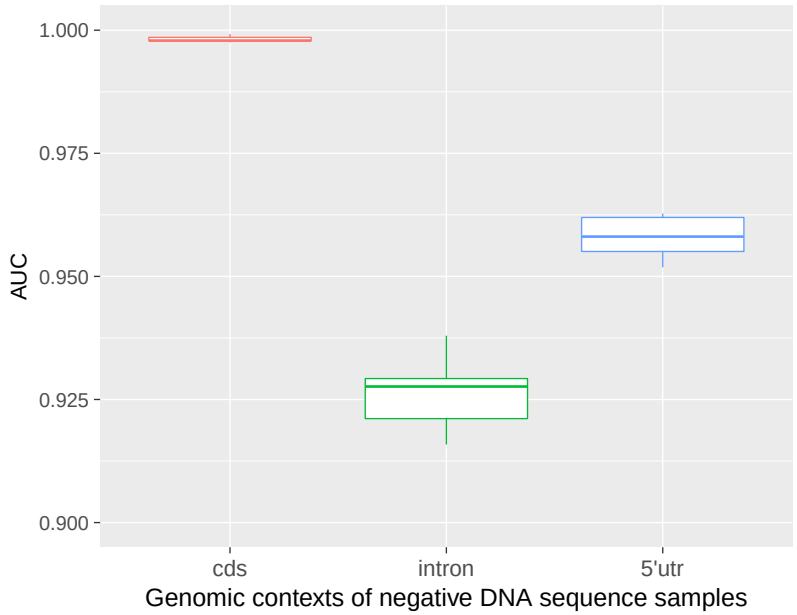


Figure 4.8 Prediction performance of DeepPolyA for alternative genomic contexts of negative DNA sequence samples.

the alignment-based method, we accumulate the motifs for the total 16 filters in DeepPolyA and assess them through comparing against JASPAR database [71], which has abundant known motifs and is widely applied as standard representation of transcription factor DNA-binding preferences. We utilize a tool named TOMTOM with predefined statistical measure of motif-motif similarity [72, 73] to compare the motifs detected by DeepPolyA with those in JASPAR database and present an alignment for each pair of considerable matches. Our visualization results of extracted motifs and comparison with JASPAR are shown in Figure 4.9. The second motif among them has the similar format with "AATAAA", which is a significant upstream element signal for poly(A) sites. Moreover, Figure 4.10 illustrates the saliency map [74] visualization of the entire testing DNA sequence. We can discover that the signal is strong around the poly(A) site. As shown in Figure 4.10, the strongest signal is at position 131 nt, which is exactly the position of the target poly(A) sites in positive samples.

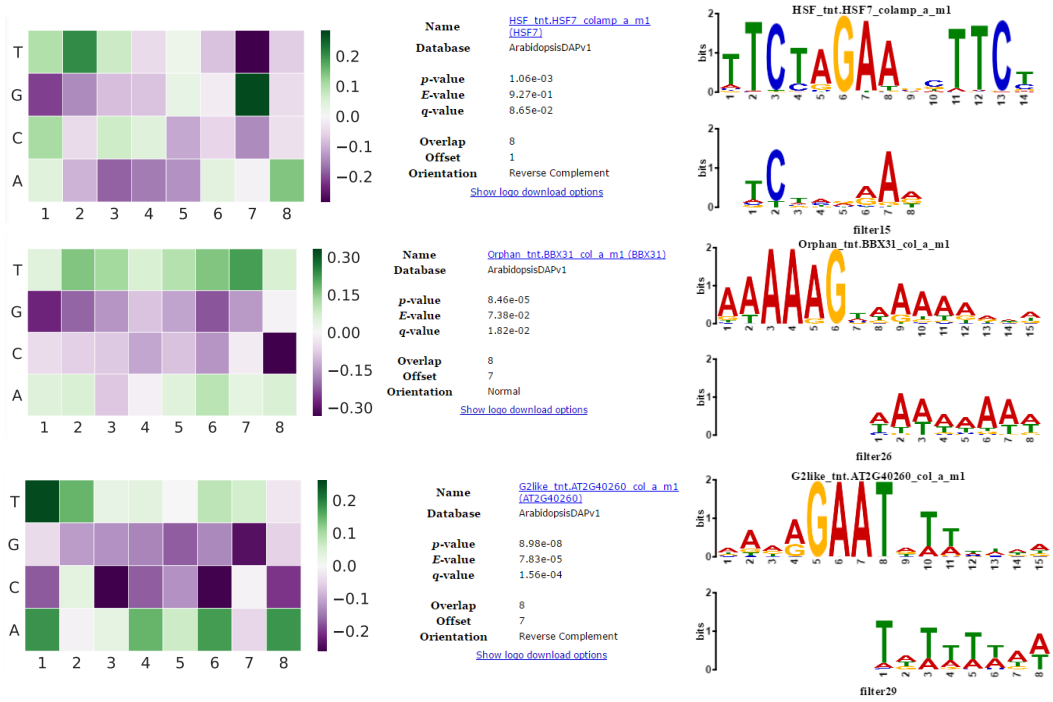


Figure 4.9 Three convolution kernels visualized from JASPAR using TOMTOM.

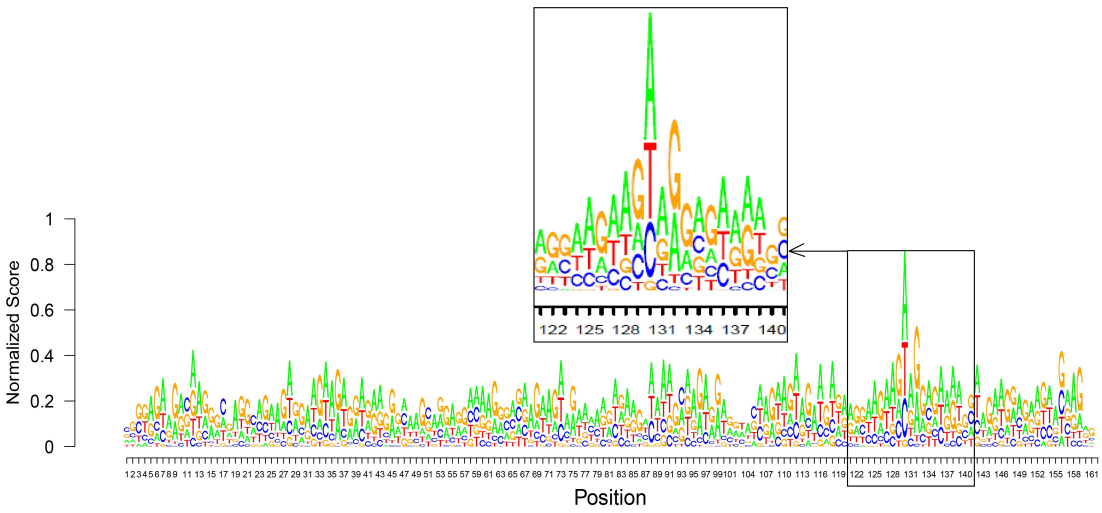


Figure 4.10 Saliency map visualization of an entire sequence with a zoom-in view of the sites around poly(A) sites.

4.5 Conclusion

In this chapter, we proposed DeepPolyA, a deep convolutional neural network approach, to automatically and accurately modeling the poly(A) signals. Using the plant Arabidopsis thaliana gene sequences datasets with one-hot encoding method, we trained several competing deep learning models with various architectures and compared the classification performance with baseline machine learning methods through several significant metrics. The evaluation results show that DeepPolyA outperforms all the competing methods without involving extensive manual feature engineering. We visualized the learned motifs of the first convolutional layer using TOMTOM against the JASPAR motif datasets to demonstrate that DeepPolyA can automatically extract poly(A) signals and features from the raw sequence data.

CHAPTER 5

TRNA-DL: A DEEP LEARNING APPROACH TO IMPROVE TRNASCAN-SE PREDICTION RESULTS

5.1 Introduction

tRNAscan-SE is the leading tool for tRNA annotation that has been widely used in the field. However, tRNAscan-SE can return a significant number of false positives when applied to large sequences. Recently, conventional machine learning methods have been proposed to address this issue, but their efficiency can be still limited due to their dependency on handcrafted features. With the growing availability of large-scale genomic datasets, deep learning methods, especially convolutional neural networks, have demonstrated excellent power in characterizing sequence pattern in genomic sequences. Thus, we hypothesize that deep learning may bring further improvement for tRNA prediction.

This chapter proposes a new computational approach based on deep neural networks, to predict tRNA gene sequences. We design and investigate various deep neural network architectures. We use tRNA sequences as positive samples and the false positive tRNA sequences predicted by tRNAscan-SE in coding sequences as negative samples, to train and evaluate the proposed models by comparison with the conventional machine learning methods and popular tRNA prediction tools. Using one-hot encoding method, our proposed models can extract features without involving extensive manual feature engineering and can outperform the existing methods under different performance metrics. The proposed deep learning methods can reduce the false positives output by the state-of-art tool tRNAscan-SE substantially. Coupled with tRNAscan-SE, it can serve as a useful complementary tool for tRNA annotation. The application to tRNA prediction demonstrates the superiority of deep learning in automatic feature generation for characterizing sequence patterns.

5.2 Motivation and Related Work

Transfer RNA (tRNA) plays a central in protein translation [17]. Thus, accurate prediction and annotation of tRNA are fundamental for interpreting and improving protein translation in all living cells. The most popular computational algorithm for tRNA prediction tRNAscan can be dated back more than 20 years ago [18]. It develops a seminal covariance model to describe the secondary structural profile and primary sequence consensus of an RNA sequence. The covariance models essentially are probabilistic secondary structure profiles based on stochastic context-free grammars. Since then, quite a few improved software tools have been developed for tRNA annotation and prediction, such as tRNAscan-SE [75, 76], ARAGORN [77] and ARWEN [78]. Based on homology with recognized tRNA consensus sequences, ARAGORN develops heuristic algorithms to search *in silico* for tRNA genes and predict tRNA secondary structure as a cloverleaf diagram [77]. ARWEN focuses on detecting tRNAs from metazoan mitochondrial nucleotide sequences, which are usually degenerate in sequence and structure compared with tRNAs in their bacterial ancestors. It employs a heuristic algorithm that searches for specific hairpin structures [78]. The tRNAscan-SE represents a practical application of RNA covariance models, which can detect a remarkable number of tRNAs in DNA sequences with good accuracies. After extracting tRNAs candidate segments from the input using a modified version of tRNAscan and a Pavesi algorithm [79], tRNAscan-SE passes them to the covariance models [18] for predicting secondary structural profile of tRNAs. Due to its superior performance, tRNAscan-SE remains the *de facto* tool for identifying tRNA genes encoded in genomes [75]. More recently, tRNAscan-SE 2.0 [76] is released, which employs improved covariance model search technology enabled by the Infernal 1.1 software [80].

Although the sensitivity of the covariance model is high in detecting tRNAs, it is not powerful enough to separate tRNAs from pseudo-tRNAs. As a result, a significant

number of false positives can be still output from tRNAscan-SE. To address this problem, Zou et al. propose a machine learning-based method and build an ensemble classifier LibMutil to improve the tRNAscan-SE annotations and predictions [81]. They use all the true tRNAs as the positive samples and pseudo tRNAs predicted from coding region sequences by tRNAscan-SE as the negative samples for training their model. Then they employed a three feature-extraction method to construct a compressed feature set consisting of local sequence, structure-sequence, and multilevel features. Using these features, they constructed an ensemble method for classification consisting of eight common classifiers, namely AdaBoost.M, Bagging, Naive Bayes, Logistic, Random Forest, Random Tree, J48, and KNN. They showed that their method could improve the tRNAscan-SE prediction results substantially. However, we note that their method required in-depth domain knowledge as it involved extensive manual feature engineering in the feature extraction and construction step. In addition, the loss of each nucleotide's essential position information in the sequence would affect the performance of prediction. Thus, in this work, we propose to use the sequences themselves as direct inputs for building a discriminative model to distinguish functional tRNAs from pseudo-tRNAs using a training set comprising all the acknowledged true and pseudo-tRNAs. Deep learning models are the candidates for our consideration.

With the growing availability of large-scale genomic datasets and advanced computational techniques, deep learning-based methods have been proposed to automatically identify and understand regulatory regions of the genome directly from DNA sequences, and predict the profile of unknown sequences based on learned knowledge [27]. Amongst a set of deep neural networks, convolutional neural networks (CNNs), have been seen in many applications in analysis of genomic sequences, for example, motif discovery [22], DNA binding site prediction [65], HLA class I-peptide

binding prediction [64], functional effect prediction of noncoding variants [24, 25], among others [23, 66].

In this chapter, we propose a computational method based on deep neural networks for predicting tRNA gene sequences. Our main contributions are summarized as follows:

1. We propose a deep learning approach tRNA-DL [1] to separate tRNA from pseudo-tRNAs. To the best of our knowledge, this is the first application of deep learning to this problem.

2. We show in this chapter that the proposed model could automatically learn predictive tRNA-related features and patterns in a data-driven manner without involving extensive manual feature engineering. The model first obtains low-layer features from DNA sequences via lower convolutional layers, and then forms high-level, complex features through the upper nonlinear transformation layers.

3. We investigate various deep learning architectures extensively to evaluate their performance in predicting tRNA.

4. We show that the proposed model, without manual feature engineering, can outperform conventional machine learning models including LibMutil, Support Vector Machine, Adaboost, KNN and Random Forest. It improves the prediction results by the tRNAscan-SE substantially.

Classical machine learning methods require researchers' prior knowledge and experience to predefine and cultivate features by counting or summarizing known genomic patterns (e.g., regulatory variants, k-mer, and structural elements). In contrast, our method can automatically learn the problem-related knowledge and

[1]Codes are freely available on Github: https://github.com/stella-gao/trna-dl

extract high-level features from the raw sequence data. They can capture nonlinear dependencies and interactions among the detected patterns and features at multiple genomic scales. It is challenging and laborious for even experienced researchers to design a schema of features including all potential interactions of different sub-signals and low-level features. Thus, the deep learning methods can yield better performance as demonstrated in the real tRNA applications.

5.3 Architecture of TRNA-DL Model

To find which neural network model works the best for the tRNA prediction task, we examine several different types of models. Based on popular deep learning architectures, we build the models of three kinds: convolutional neural network (CNN), recurrent neural network (RNN) and a hybrid combination of convolutional neural network and recurrent neural network (CNN-RNN). Table 5.1 show the abbreviations of different types of deep neural network layers. Conv1D, Conv2D, MaxPooling1D, Maxpooling2D, AvgPooling1D and AvgPooling2D layers are deployed for building CNNs. LSTM, GRU, BDLSTM, and DBGRU layers are deployed for building RNNs. FC layer can be used in both CNNs and RNNs.

To find out which deep neural network architecture can achieve satisfying performance results, we construct thirteen different deep neural network architectures. The deployment and connections between layers are shown in Table 5.2, with abbreviations listed. Among all these models, the CNNs are CF, CCMF, CMCMF, CCMCAF, CMCMCMF, CCCMF, CMCMCF2 and CCCMF2. CMCMCF2 and CCCMF2 applied two-dimensional convolutional neural networks while the others applied one-dimensional neural networks. A two-dimensional convolution can be considered as sliding one function on top of another, multiplying and adding. Although mostly they are used for image processing, they can also be used in feature extraction from genomic sequences, such as DeepSEA [24]. Since RNNs are seldom

Table 5.1 Abbreviations of Different Types of Deep Neural Network Layers

Abbreviation	**Type of Layers**
Conv1D	One-dimensional convolutional layer
Conv2D	Two-dimensional convolutional layer
MaxPool1D	Max pooling layer for temporal data
MaxPool2D	Max pooling layer for spatial data
AvgPool1D	Average pooling for temporal data
AvgPool2D	Average pooling for spatial data
FC	Fully connected layer
LSTM	Long-short term memory layer
GRU	Gated recurrent unit layer
BDLSTM	Bidirectional wrapper for LSTM layer
BDGRU	Bidirectional wrapper for GRU layer

used alone for training gene sequences. To address it, we build one RNN model LLLF for evaluation comparison with other models. The leftover four architectures CMBLF, CCMBLF, CMBGF, and CCMBGF belong to CNN-RNN architecture, which we get the idea from DanQ [25].

For the tRNA prediction problem, we would select the model that demonstrates the best prediction potential among the above architectures. The following experiments section shows that CCMBLF performs best among the thirteen different architectures of models. Thus, we select CCMBLF as our proposed model, named as tRNA-DL. The proposed model tRNA-DL is shown in Figure 5.1, which consists of two convolutional layers and one max-pooling layer to identify predictive motifs from the context of DNA sequences and one bidirectional LSTM layer, one fully connected hidden layer with a ReLU activation function to model motif interactions [27].

With one-hot encoding method, we can preserve the vital position information of each nucleotide in DNA sequences. Because we read each entire sequence as input to the model, we can preserve all the position of each nucleotide without missing the relationship information between any two nucleotides. Table 5.3 shows the proposed architecture with the hyperparameters setting based on the performance evaluation on a validation dataset. The size column describes the kernel size of the convolutional layer, the window size of the max-pooling layer and the size of the fully connected layer. We set the number of kernels as 16 and 64 in the first and second convolutional layers, respectively. We first utilize a convolutional layer that directly operating on the encoded bit matrix to scan the motif. Next, we apply a max-pooling layer subsequently to reduce dimensionality and a second convolutional layer to model the interactions between motifs generated by previous layers, where the outputs of the convolutional layers are then flattened into vectors, fed to the bidirectional LSTM and the fully connected layer for further extraction. The final results are then converted into probabilities via a "sigmoid" function.

Table 5.2 Abbreviations of the Deep Neural Network Models

No.	Abbreviation: Prediction Deep Learning Model Architectures
1	CF: Conv1D + FC + SGD
2	CCMF: Conv1D + Conv1D + MaxPool1D + FC + SGD
3	CMCMF: Conv1D + MaxPool1D + Conv1D + MaxPool1D + FC + SGD
4	CCMCAF: Conv1D + Conv1D + MaxPool1D + Conv1D + AvgPool1D + FC + SGD
5	CMCMCMF: Conv1D + MaxPool1D + Conv1D + MaxPool1D + Conv1D + MaxPool1D + FC + SGD
6	CCCMF: Conv1D + Conv1D + Conv1D + MaxPool1D + FC + SGD
7	CMCMCF2: Conv2D + MaxPool2D + Conv2D + MaxPool2D + Conv2D + FC + SGD
8	CCCMF2: Conv2D + Conv2D + Conv2D + MaxPool2D + FC + SGD
9	LLLF: LSTM + LSTM + LSTM + FC + SGD
10	CMBLF: Conv1D + MaxPool1D + BDLSTM + FC + RMSprop
11	CCMBLF: Conv1D + Conv1D + MaxPool1D + BDLSTM + FC + RMSprop
12	CMBGF: Conv1D + MaxPool1D + BDGRU + FC + RMSprop
13	CCMBGF: Conv1D + Conv1D + MaxPool1D + BDGRU + FC + RMSprop

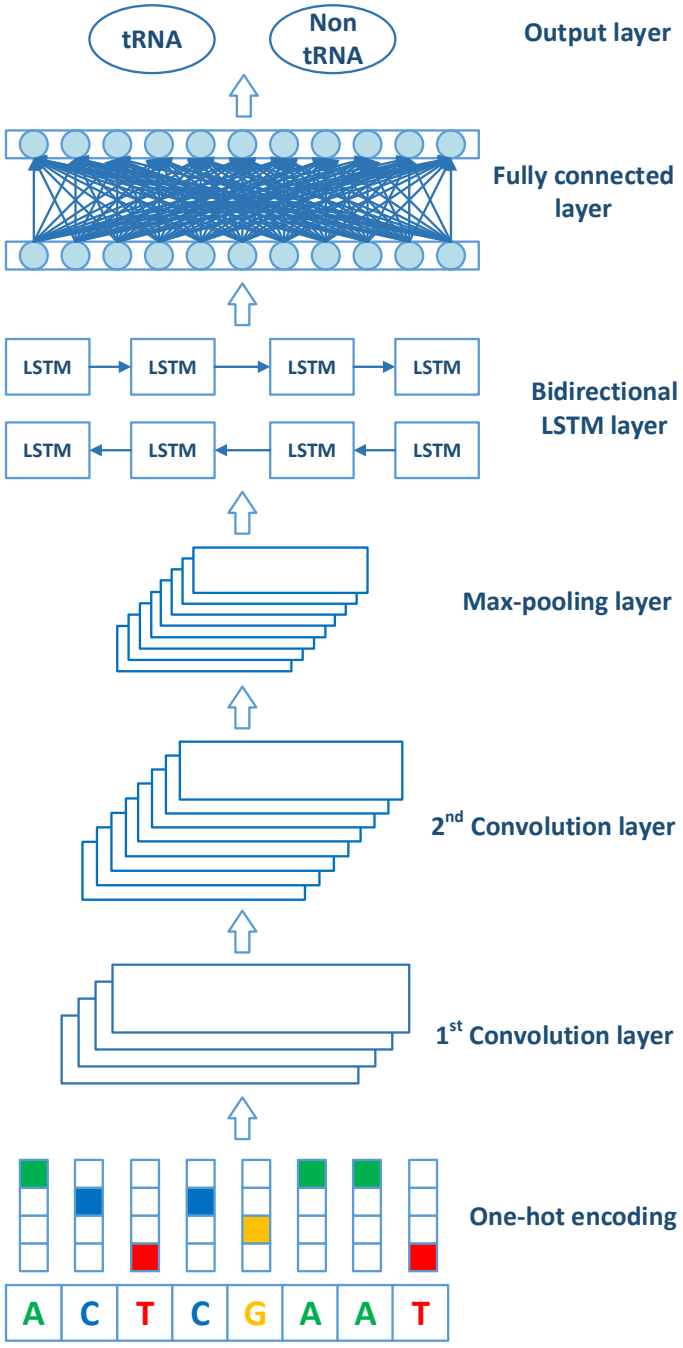


Figure 5.1 Deep learning model chosen with the best performance on the validation set.

Table 5.3 tRNA-DL Model Architecture and Hyperparameters

Layer No.	**Layer Type**	**Size**	**Output**
0	INPUT	-	4×134
1	CONV	16×4×4	16×131
2	RELU	-	16×131
3	DROPOUT	-	16×131
4	CONV	64×4×4	64×128
5	RELU	-	64×128
6	POOL	4×2	64×64
7	BDLSTM	64×4×6	128×64
8	RELU	-	128×64
9	FC	128	128
10	DROPOUT	-	128
11	FC	1	1
12	SIGMOID	1	1

5.4 Experiments and Results

Here, we will illustrate data processing, the libraries and packages used for training models, the experimental environment as well as the numerical evaluation performance of the proposed model compared with both the deep learning models and baseline classical machine learning models.

5.4.1 Data Source and Data Preprocessing

Our goal is to further improve the tRNAscan-SE software [75] tRNA prediction results with our proposed tRNA-DL deep learning model. Deep Learning models mostly rely on the training dataset. Choosing a high-quality negative dataset for our model is significant. Following Zou et al. [81], we adopt the same positive and negative samples as those used in [81]. We choose the false positive tRNA sequences as negative samples. The same tRNA sequences datasets consisting of both positive and negative samples chosen from the baseline machine learning method-based literature [81] were utilized for our experiments. The lengths of selected tRNA sequences are in the range of [54, 134]. Positive samples have 623 tRNA sequences of 104 species, which are the sequences obtained by randomly sampling from tRNAdb 2009 database [82]. The database has over 12,000 tRNA gene sequences of 577 species in total. Negative samples are obtained by randomly sampling from the predicted false positive tRNA sequences through tRNAscan-SE in the EMBL CDS database [83], which consists of known and predicted coding sequences. The extracted negative samples contain 1183 false positive tRNA sequences, where the redundant sequences have been discarded through CD-HIT software with a parameter c=0.9 [84].

We divide the entire dataset into training, validation and test sets and make sure there is no overlap between them. The ratio to split them is 7:2:1. We use the training and validation datasets to select the best performance deep neural networks model. The test dataset is used later for prediction performance comparison against baseline

machine learning methods and existing programs and tools. Note that the validation dataset is then partitioned to a sub-validation dataset and a sub-test dataset for evaluating the performance of different deep learning models. During the training process, the sub-validation dataset helps tune the weights and parameters, which the training dataset learns. Later, the sub-test dataset can show the performance results based on this 90% of entire datasets, and then the best model among them is selected. The test dataset is adopted to evaluate the performance of our selected model tRNA-DL against baseline machine learning methods and other prediction tools.

5.4.2 Experiments on Deep Learning Model Selection

We applied Hyperopt [39] package to tune the hyperparameters for our models. We first randomly picked five to six values in a large range with same interval between each other and then narrowed down the range as much as possible. We repeated this way to finally find the hyperparameters that were suitable for our models. Figures 5.2 and 5.3 illustrate the loss values and accuracy values respectively during model training. The model weights are updated and saved as soon as there is a better performance. The training process stops at about 265 epochs since the best weights are obtained at 65 epochs. There are no updates on the weights after 200 epochs. Then to avoid overfitting and save time, we stop the training and evaluate our models.

Figure 5.4 shows the prediction performance of all the thirteen deep neural network models mentioned above. Among all the performance assessment of the thirteen constructed deep neural network methods, we can discover that the CCMBLF model performs best, which is selected as our proposed model, named as tRNA-DL.

Figure 5.5 shows the area under receiver operating characteristics (ROC curve) and Figure 5.6 also shows the area under precision-recall curve (PR curve) for deep learning based methods. From the curves, we can find that our a hybrid combination

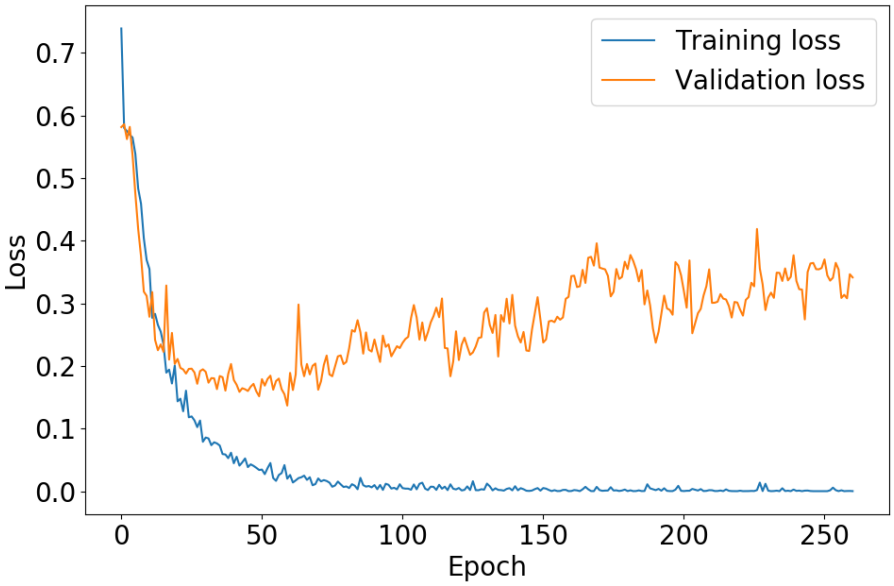


Figure 5.2 Loss value during training process.

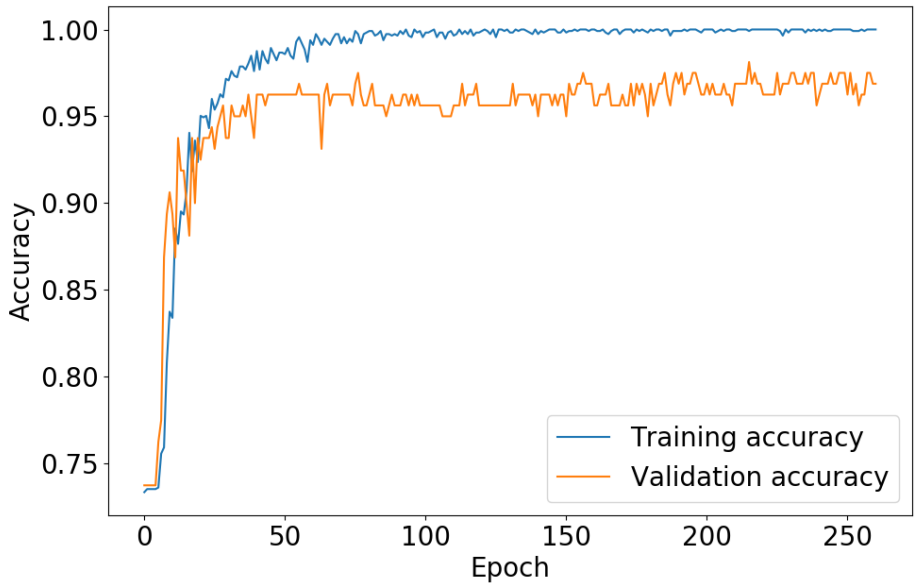


Figure 5.3 Accuracy value during training process.

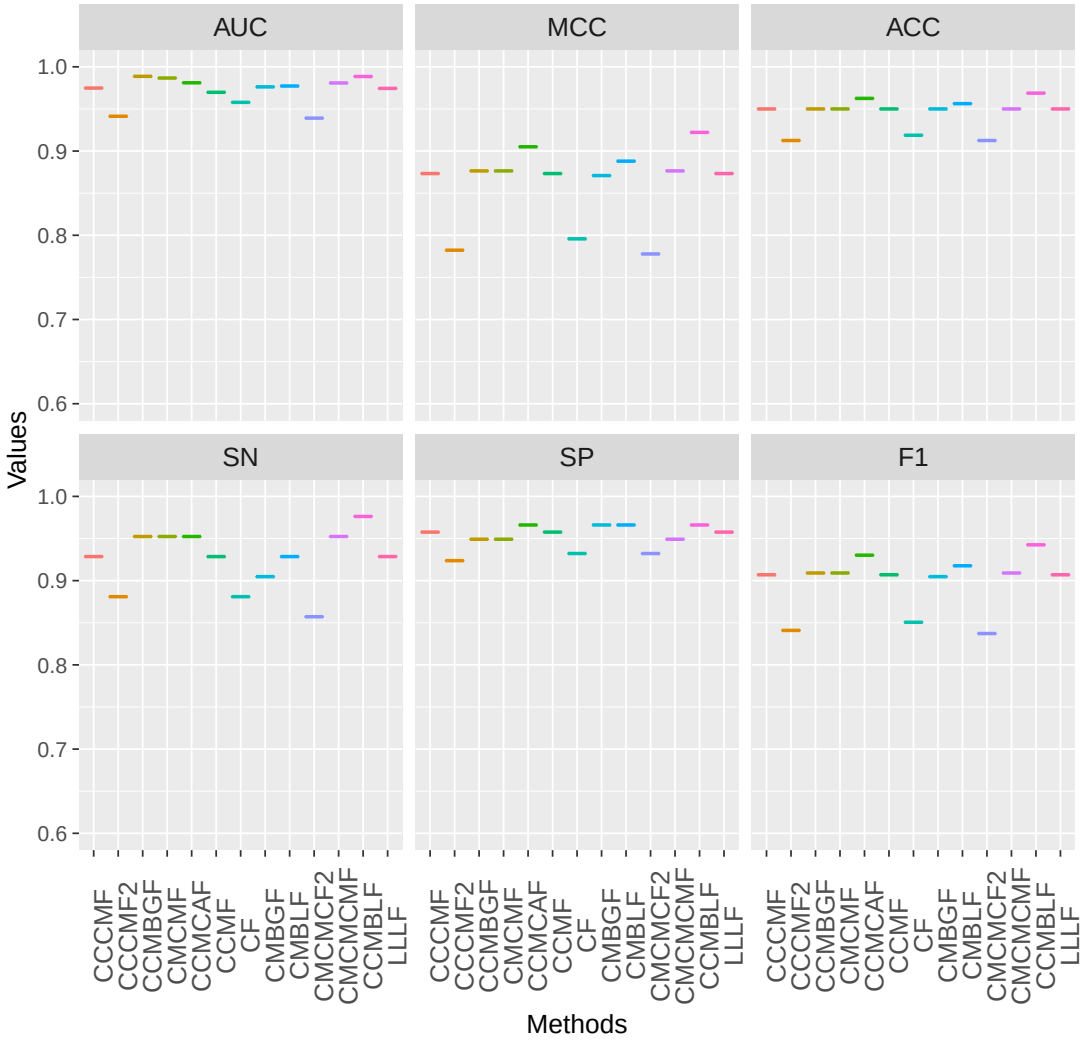


Figure 5.4 Prediction performance of various deep learning models. Performance is measured using boxplot by six evaluation metrics, including the area under the receiver-operating characteristic curve (AUC), accuracy (ACC), recall (Sn), specificity (Sp), Matthew's correlation coefficient (MCC), and F-score (F1).

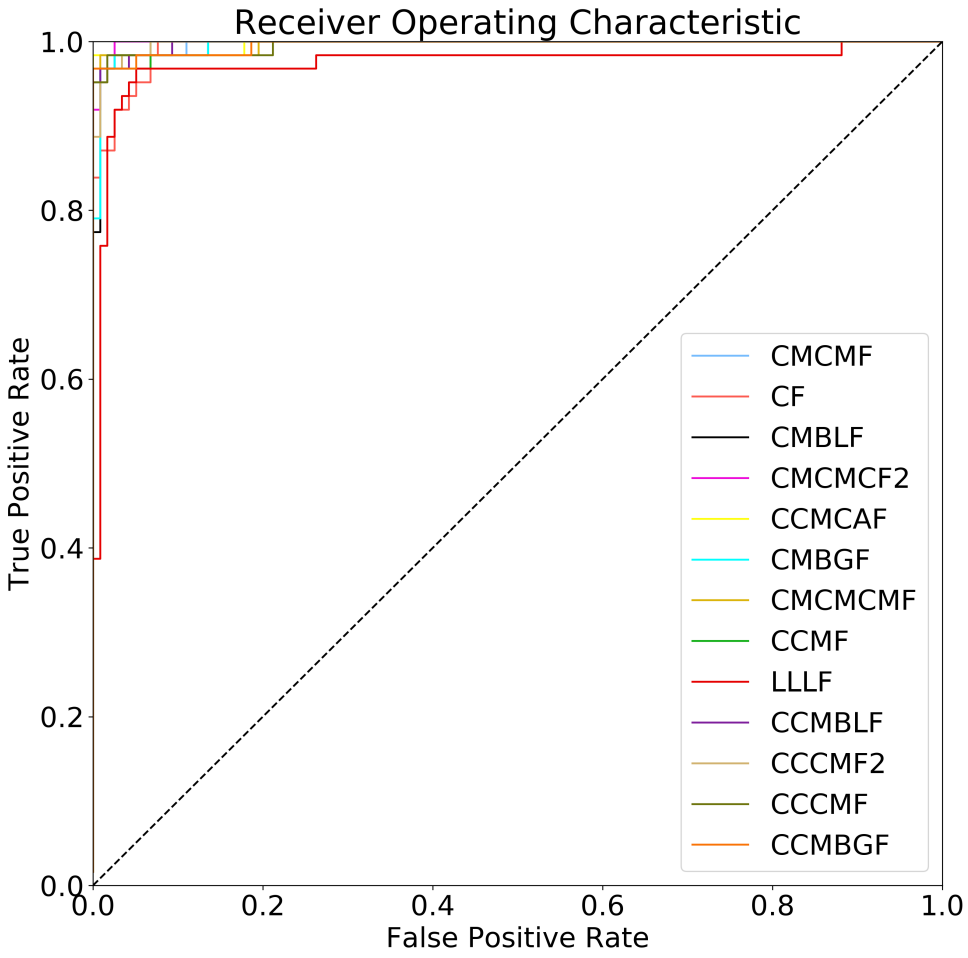


Figure 5.5 ROC curves of the prediction performance among all the deep learning methods.

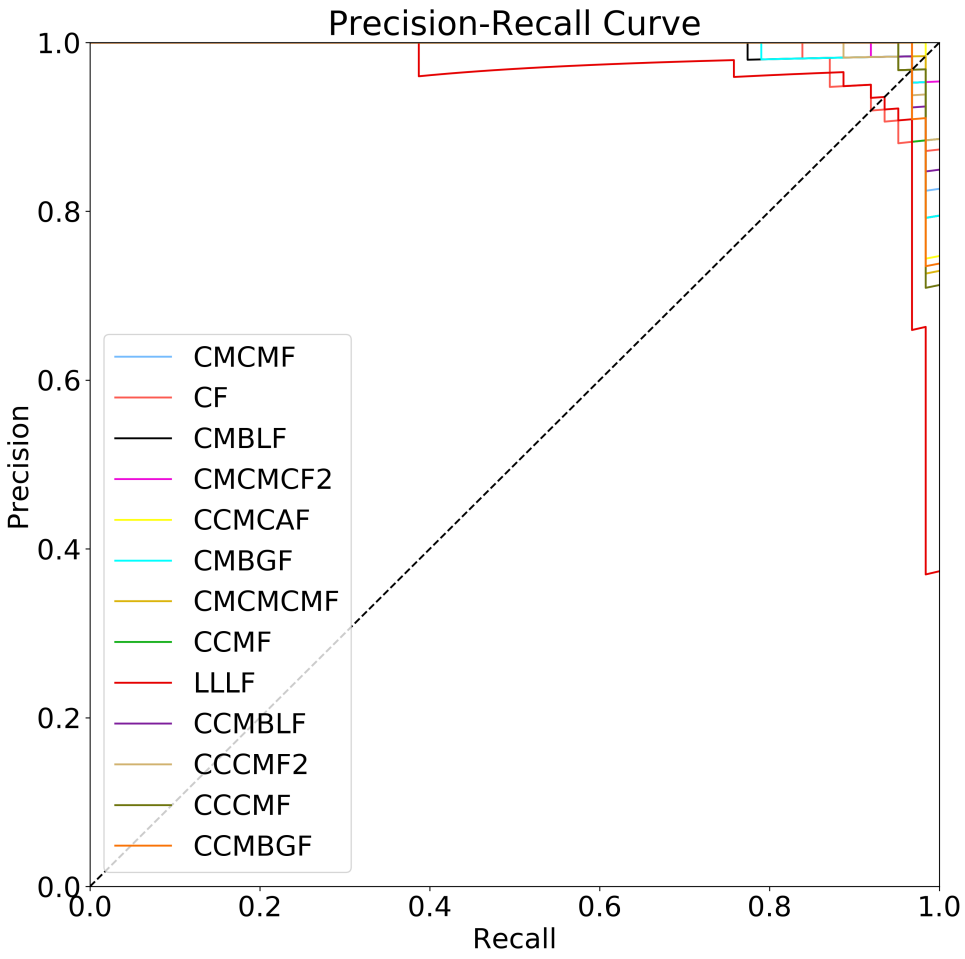


Figure 5.6 Precision-Recall curves of the prediction performance among all the deep learning methods.

of CNN and RNN model demonstrates the best performance. The CNN layers are used to extract sequence patterns which are related to the vital position information. The RNN layers, especially bidirectional LSTM layers are applied on top of the CNN layers so as to extract the inner relation between adjacent nucleotides or even between the beginning part and ending part of the entire sequence. Figure 5.7 illustrates the auc values of all the thirteen deep neural network models and also the classical machine learning models. It is clear that our model tRNA-DL obtains a higher auc value than other models. From that, we can find that the hybrid combination of CNN and RNN models demonstrate the best performance and can always get better performance than CNN models. The models that are using bidirectional LSTM layers perform better than the models using one-directional LSTM layers, one-directional GRU layers or bidirectional GRU layers.

5.4.3 Comparison with Existing Machine Learning Methods

We compare our proposed method tRNA-DL with the state-of-the-art machine learning methods, including Support Vector Machine (SVM) algorithm with a linear kernel, Random Forest (RF) algorithm with 50 trees, K-nearest neighbors (KNN) algorithm with K=4, AdaBoost algorithm with a J48 base classifier and a LibMutil ensemble classifier [81]. All the machine learning methods are implemented in Weka [68] software. As shown in Table 5.4, our selected deep learning model tRNA-DL achieves remarkable and stable prediction performance on the test dataset, which outperforms all the classical machine learning methods (including the baseline method LibMutil) among all metrics.

5.4.4 Comparison with Existing Softwares and Tools

To validate our proposed model tRNA-DL and evaluate its effectiveness, we compare the performance results with three other tools, which are tRNAscan-SE, ARWEN and ARAGORN. The results based on six evaluation metrics are shown in Table 5.5.

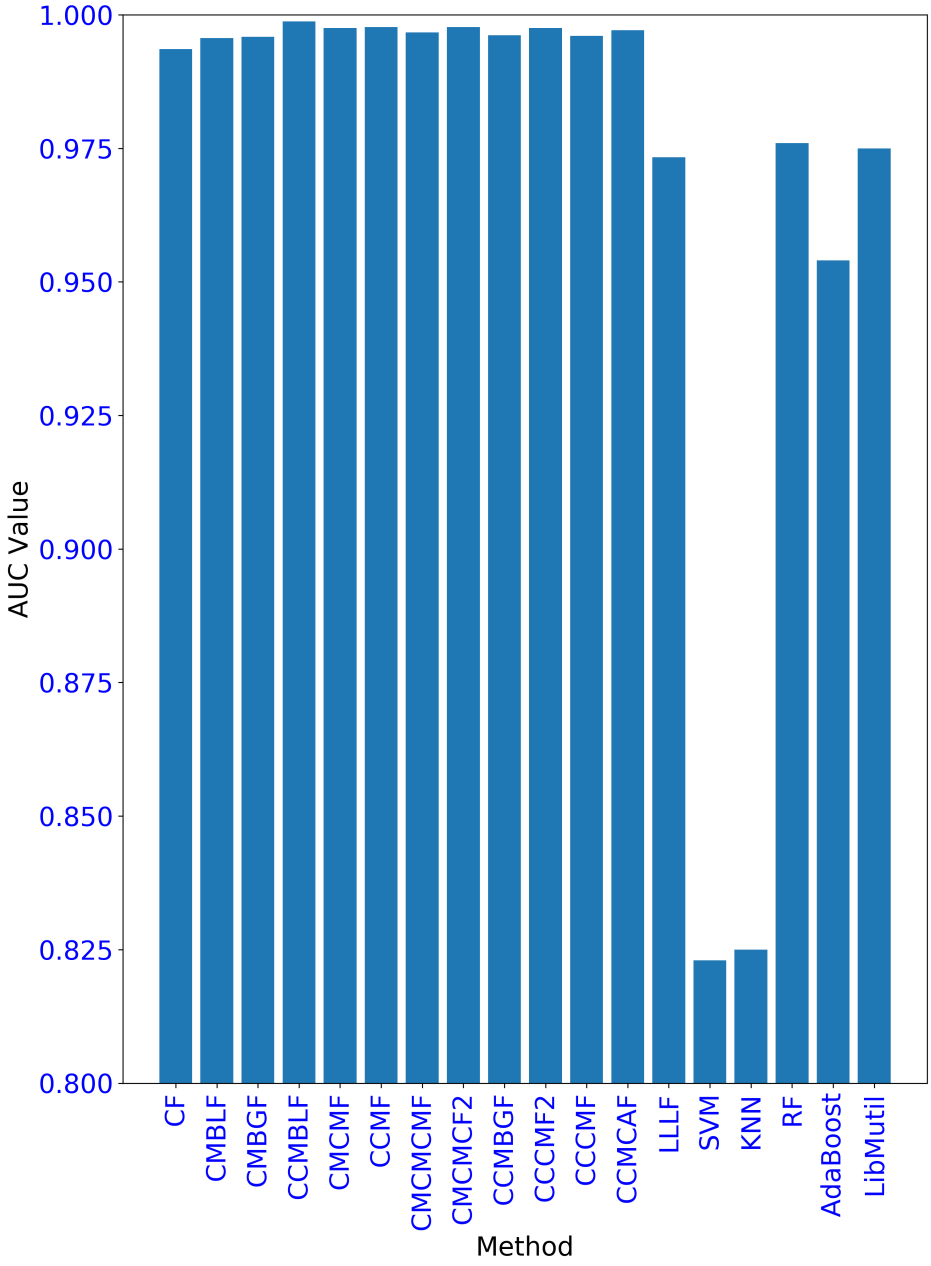


Figure 5.7 AUC values for all models.

Table 5.4 Prediction Performance of tRNA-DL Comparing with Existing Machine Learning Methods

Metrics	**SVM (linear)**	**KNN (K=4)**	**RF (N=50)**	**AdaBoost (J48)**	**Lib-Mutil**	**tRNA-DL**
AUC	82.3%	82.5%	97.6%	95.4%	97.5%	**99.8%**
MCC	63.7%	44.2%	75.7%	77.9%	81.1%	**96.4%**
ACC	84.3%	74.7%	89.8%	90.7%	92.0%	**98.4%**
SN	84.3%	74.7%	89.8%	90.7%	92.0%	**96.8%**
SP	80.2%	71.5%	80.0%	84.8%	88.7%	**99.2%**
F1	84.4%	75.3%	89.4%	90.6%	92.0%	**97.6%**

We can discover that tRNA-DL dramatically outperforms the other three tools when it comes across the false positive tRNA sequences. Overall, the results indicate that our model tRNA-DL can detect the false positive tRNA sequence and help improve the tRNAscan-SE tool prediction results a lot.

5.5 Conclusion

In this chapter, because of the unacceptable false positive rate of tRNAscan-SE for a significant amount of sequences, we introduce a deep learning approach to predict tRNA so as to further improve tRNAscan-SE annotation results. The benefit of introducing the deep learning approach is to automatically and accurately extract satisfactory features without involving extensive manual feature engineering for better prediction performance. We obtain positive samples from tRNAdb 2009 and obtain negative samples that are false positive tRNA sequences predicted by tRNAscan-SE in coding sequences. The positive and negative samples we used are the same as those used in [81]. After removing the redundant data, we transfer the tRNA

Table 5.5 Prediction Performance of tRNA-DL Comparing with Existing tRNA Prediction Tools

Metrics	**tRNAscan-SE**	**ARWEN**	**ARAGORN**	**tRNA-DL**
AUC	53.1%	52.1%	57.7%	**99.8%**
MCC	10.4%	10.0%	20.6%	**96.4%**
ACC	40.1%	37.9%	46.2%	**98.4%**
SN	95.2%	**98.4%**	95.2%	96.8%
SP	10.9%	5.9%	20.2%	**99.2%**
F1	52.4%	52.3%	52.4%	**97.6%**

genomic sequences into encoded bit matrices using one-hot encoding method. We design thirteen competing deep learning models of various architectures, train these models and choose the one with the best evaluation performance as our model. We evaluate our model based on the test dataset and compare the classification performance with baseline machine learning methods through several meaningful metrics. The evaluation results demonstrate that the chosen model outperforms the competing machine learning methods as measured by its higher accuracy and AUC score. In summary, the proposed deep learning method is a promising tool for tRNA classification and prediction. It shows a high potential for pursuing this approach to solve other related problems in computational biology, which will be considered in our follow-up studies.

CHAPTER 6

CONCLUSION

This dissertation focuses on the development of deep learning methods for identifying genomic sequences and mining sequence patterns. To our knowledge, we are the first to compare different deep learning models for the sequence pattern recognition problem. The main contributions of this dissertation are as listed below.

First, motivated by the investigated deep learning methods for genomic sequences identification, we will focus on pattern recognition based on self-generated simulated sequence datasets with designed patterns. There are two major kinds of deep learning models, including convolutional neural network and recurrent neural network. Convolutional Neural Network and Recurrent Neural Network models are appropriate for different types of sequence patterns. In view of the simulation results, we believe that we can always find an appropriate deep learning model for a specific sequence pattern. In addition, a saliency-map-based method is applied to visualize the learned sequence patterns. We also study different deep learning approached using real genomic sequence data, and demonstrate that a convolutional neural network model is slightly better than a recurrent neural network model for mining gene sequence patterns. Through the visualization based on the saliency map, we can find the convolutional neural network can find out the primary gene pattern: start codon and stop codon.

Second, a CNN-based model named DeepPolyA is proposed to predict poly(A) site from the plant Arabidopsis thaliana gene sequences, which is the first deep learning based approach to this research issue. DeepPolyA could automatically learn poly(A)-related motifs without involving any manual feature engineering. DeepPolyA outperforms not only the conventional machine learning methods including Support

Vector Machine (SVM), Bayesian Networks and Random Forest, but also the existing popular deep learning models.

Third, a deep learning approach tRNA-DL is proposed to separate tRNA from pseudo-tRNAs. It shows that without manual feature engineering, tRNA-DL can outperform conventional machine learning models including Support Vector Machine, Adaboost, KNN, Random Forest and an ensemble model of all these models. It improves the prediction results by the tRNAscan-SE substantially. Coupled with tRNAscan-SE, it can serve as a useful complementary tool for tRNA annotation. The above applications demonstrate the superiority of deep learning in automatic feature generation for characterizing sequence patterns.

In summary, comparing to traditional machine learning methods, deep learning model can be generally applied for successfully recognizing various genomic sequence pattern and can always achieve better performance. Convolutional neural network and recurrent neural network are good at detecting different kinds of sequence patterns. They are comparable to each other, the integration of both is even better. For mining genomic patterns, CNN is usually a right way to start. Sometimes, RNN can be attached CNN to help CNN get better results. There are still some limitations in current research. For example, other recent variants of RNNs have not been explored. For some patterns, especially with non-fixed positions, saliency map didn't highlight pattern positions well. In our future work, we would explore recent and modern RNN architectures to improve the performance further. We would also investigate how to locate the pattern positions and summarize the patterns after the sequences with the pattern have been identified so that we can improve over the saliency map.

REFERENCES

[1] Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2625–2634, 2015.

[2] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057, 2015.

[3] Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3128–3137, 2015.

[4] Jiang Wang, Yi Yang, Junhua Mao, Zhiheng Huang, Chang Huang, and Wei Xu. Cnn-rnn: A unified framework for multi-label image classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2285–2294, 2016.

[5] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen AWM van der Laak, Bram Van Ginneken, and Clara I Sánchez. A survey on deep learning in medical image analysis. *Medical image analysis*, 42:60–88, 2017.

[6] Duyu Tang, Bing Qin, and Ting Liu. Document modeling with gated recurrent neural network for sentiment classification. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, pages 1422–1432, 2015.

[7] Ankit Kumar, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, and Richard Socher. Ask me anything: Dynamic memory networks for natural language processing. In *International Conference on Machine Learning*, pages 1378–1387, 2016.

[8] William Chan, Navdeep Jaitly, Quoc Le, and Oriol Vinyals. Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 4960–4964. IEEE, 2016.

[9] Xiao-Nan Fan and Shao-Wu Zhang. lncrna-mfdl: identification of human long non-coding rnas by fusing multiple features and using deep learning. *Molecular BioSystems*, 11(3):892–897, 2015.

[10] Rashmi Tripathi, Sunil Patel, Vandana Kumari, Pavan Chakraborty, and Pritish Kumar Varadwaj. Deeplnc, a long non-coding rna prediction tool using deep neural network. *Network Modeling Analysis in Health Informatics and Bioinformatics*, 5(1):21, 2016.

[11] Byunghan Lee, Junghwan Baek, Seunghyun Park, and Sungroh Yoon. deeptarget: end-to-end learning framework for microrna target prediction using deep recurrent neural networks. In *Proceedings of the 7th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*, pages 434–442. ACM, 2016.

[12] Bin Tian and James L Manley. Alternative polyadenylation of mrna precursors. *Nature Reviews Molecular Cell Biology*, 2016.

[13] Ran Elkon, Alejandro P Ugalde, and Reuven Agami. Alternative cleavage and polyadenylation: extent, regulation and function. *Nature Reviews Genetics*, 14(7):496–506, 2013.

[14] Denghui Xing and Qingshun Quinn Li. Alternative polyadenylation and gene expression regulation in plants. *Wiley Interdisciplinary Reviews: RNA*, 2(3):445–458, 2011.

[15] Shanxin Zhang, Jiuqiang Han, Jun Liu, Jiguang Zheng, and Ruiling Liu. An improved poly(a) motifs recognition method based on decision level fusion. *Computational biology and chemistry*, 54:49–56, 2015.

[16] Yuanhang Liu, Ping Wu, Jingqi Zhou, Teresa L Johnson-Pais, Zhao Lai, Wasim H Chowdhury, Ronald Rodriguez, and Yidong Chen. Xbseq2: a fast and accurate quantification of differential expression and differential polyadenylation. *BMC bioinformatics*, 18(11):384, 2017.

[17] Mathias Sprinzl, Thomas Hartmann, Joachim Weber, Jutta Blank, and Robert Zeidler. Compilation of trna sequences and sequences of trna genes. *Nucleic acids research*, 17(Suppl):r1, 1989.

[18] Sean R Eddy and Richard Durbin. Rna sequence analysis using covariance models. *Nucleic acids research*, 22(11):2079–2088, 1994.

[19] S Clancy et al. Chemical structure of rna. *Nature Education*, 1(1):223, 2008.

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[21] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

[22] Babak Alipanahi, Andrew Delong, Matthew T Weirauch, and Brendan J Frey. Predicting the sequence specificities of dna-and rna-binding proteins by deep learning. *Nature biotechnology*, 33(8):831–838, 2015.

[23] David R Kelley, Jasper Snoek, and John L Rinn. Basset: learning the regulatory code of the accessible genome with deep convolutional neural networks. *Genome research*, 26(7):990–999, 2016.

[24] Jian Zhou and Olga G Troyanskaya. Predicting effects of noncoding variants with deep learning-based sequence model. *Nature methods*, 12(10):931–934, 2015.

[25] Daniel Quang and Xiaohui Xie. Danq: a hybrid convolutional and recurrent deep neural network for quantifying the function of dna sequences. *Nucleic acids research*, 44(11):e107–e107, 2016.

[26] Xueliang Liu. Deep recurrent neural network for protein function prediction from sequence. *arXiv preprint arXiv:1701.08318*, 2017.

[27] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[28] Gary D Stormo, Thomas D Schneider, Larry Gold, and Andrzej Ehrenfeucht. Use of the 'perceptron' algorithm to distinguish translational initiation sites in e. coli. *Nucleic acids research*, 10(9):2997–3011, 1982.

[29] Thomas D Schneider and R Michael Stephens. Sequence logos: a new way to display consensus sequences. *Nucleic acids research*, 18(20):6097–6100, 1990.

[30] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[31] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[32] François Chollet et al. Keras. 2015.

[33] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[34] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[35] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[36] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

[37] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[38] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[39] James Bergstra and David D Cox. Hyperparameter optimization and boosting for classifying facial expressions: How good can a "null" model be? *arXiv preprint arXiv:1306.3476*, 2013.

[40] Aarne Ranta. A multilingual natural-language interface to regular expressions. In *Proceedings of the International Workshop on Finite State Methods in Natural Language Processing*, pages 79–90. Association for Computational Linguistics, 1998.

[41] Yunyao Li, Rajasekar Krishnamurthy, Sriram Raghavan, Shivakumar Vaithyanathan, and HV Jagadish. Regular expression learning for information extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 21–30. Association for Computational Linguistics, 2008.

[42] Henning Fernau. Algorithms for learning regular expressions. In *International Conference on Algorithmic Learning Theory*, pages 297–311. Springer, 2005.

[43] Henning Fernau. Algorithms for learning regular expressions from positive data. *Information and Computation*, 207(4):521–541, 2009.

[44] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[45] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[46] Aaron R Quinlan and Ira M Hall. Bedtools: a flexible suite of utilities for comparing genomic features. *Bioinformatics*, 26(6):841–842, 2010.

[47] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pages 437–478. Springer, 2012.

[48] Xiaochuan Liu, Mainul Hoque, Marc Larochelle, Jean-François Lemay, Nathan Yurko, James L Manley, François Bachand, and Bin Tian. Comparative analysis of alternative polyadenylation in s. cerevisiae and s. pombe. *Genome research*, 27(10):1685–1695, 2017.

[49] Guoli Ji, Jianti Zheng, Yingjia Shen, Xiaohui Wu, Ronghan Jiang, Yun Lin, Johnny C Loke, Kimberly M Davis, Greg J Reese, and Qingshun Quinn Li. Predictive modeling of plant messenger rna polyadenylation sites. *BMC bioinformatics*, 8(1):43, 2007.

[50] Johnny C Loke, Eric A Stahlberg, David G Strenski, Brian J Haas, Paul Chris Wood, and Qingshun Quinn Li. Compilation of mrna polyadenylation signals in arabidopsis revealed a new signal element and potential secondary structures. *Plant physiology*, 138(3):1457–1468, 2005.

[51] Qingshun Li and Arthur G Hunt. A near-upstream element in a plant polyadenylation signal consists of more than six nucleotides. *Plant molecular biology*, 28(5):927–934, 1995.

[52] JUN Hu, Carol S Lutz, Jeffrey Wilusz, and BIN Tian. Bioinformatic identification of candidate cis-regulatory elements involved in human mrna polyadenylation. *Rna*, 11(10):1485–1493, 2005.

[53] Xiaohui Wu, Man Liu, Bruce Downie, Chun Liang, Guoli Ji, Qingshun Q Li, and Arthur G Hunt. Genome-wide landscape of polyadenylation in arabidopsis provides evidence for extensive alternative polyadenylation. *Proceedings of the National Academy of Sciences*, 108(30):12533–12538, 2011.

[54] Victor Solovyev and Ramzan Umarov. Prediction of prokaryotic and eukaryotic promoters using convolutional deep learning neural networks. *arXiv preprint arXiv:1610.00121*, 2016.

[55] Malik Nadeem Akhtar, Syed Abbas Bukhari, Zeeshan Fazal, Raheel Qamar, and Ilham A Shahmuradov. Polyar, a new computer program for prediction of poly(a) sites in human sequences. *Bmc Genomics*, 11(1):646, 2010.

[56] Yiming Cheng, Robert M Miura, and Bin Tian. Prediction of mrna polyadenylation sites by support vector machine. *Bioinformatics*, 22(19):2320–2325, 2006.

[57] Tzu-Hao Chang, Li-Ching Wu, Yu-Ting Chen, Hsien-Da Huang, Baw-Jhiune Liu, Kuang-Fu Cheng, and Jorng-Tzong Horng. Characterization and prediction of mrna polyadenylation sites in human genes. *Medical & biological engineering & computing*, 49(4):463–472, 2011.

[58] Ju Youn Lee, Ijen Yeh, Ji Yeon Park, and Bin Tian. Polya_db 2: mrna polyadenylation sites in vertebrate genes. *Nucleic acids research*, 35(suppl 1):D165–D168, 2007.

[59] Bo Xie, Boris R Jankovic, Vladimir B Bajic, Le Song, and Xin Gao. poly(a) motif prediction using spectral latent features from human dna sequences. *Bioinformatics*, 29(13):i316–i325, 2013.

[60] Joel H Graber, Gregory D McAllister, and Temple F Smith. Probabilistic prediction of saccharomyces cerevisiae mrna 3'-processing sites. *Nucleic acids research*, 30(8):1851–1858, 2002.

[61] Yingjia Shen, Guoli Ji, Brian J Haas, Xiaohui Wu, Jianti Zheng, Greg J Reese, and Qingshun Quinn Li. Genome level analysis of rice mrna 3'-end processing signals and alternative polyadenylation. *Nucleic acids research*, 36(9):3150–3161, 2008.

[62] Guoli Ji, Xiaohui Wu, Yingjia Shen, Jiangyin Huang, and Qingshun Quinn Li. A classification-based prediction model of messenger rna polyadenylation sites. *Journal of theoretical biology*, 265(3):287–296, 2010.

[63] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[64] Yeeleng S Vang and Xiaohui Xie. Hla class i binding prediction via convolutional neural networks. *Bioinformatics*, page btx264, 2017.

[65] Haoyang Zeng, Matthew D Edwards, Ge Liu, and David K Gifford. Convolutional neural network architectures for predicting dna–protein binding. *Bioinformatics*, 32(12):i121–i127, 2016.

[66] Jiyun Zhou, Qin Lu, Ruifeng Xu, Lin Gui, and Hongpeng Wang. Cnnsite: Prediction of dna-binding residues in proteins using convolutional neural network with sequence features. In *Bioinformatics and Biomedicine (BIBM), 2016 IEEE International Conference on*, pages 78–85. IEEE, 2016.

[67] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[68] Geoffrey Holmes, Andrew Donkin, and Ian H Witten. Weka: A machine learning workbench. In *Intelligent Information Systems, 1994. Proceedings of the 1994 Second Australian and New Zealand Conference on*, pages 357–361. IEEE, 1994.

[69] Gavin E Crooks, Gary Hon, John-Marc Chandonia, and Steven E Brenner. Weblogo: a sequence logo generator. *Genome research*, 14(6):1188–1190, 2004.

[70] Jack Lanchantin, Ritambhara Singh, Beilun Wang, and Yanjun Qi. Deep motif dashboard: Visualizing and understanding genomic sequences using deep neural networks. *arXiv preprint arXiv:1608.03644*, 2016.

[71] Anthony Mathelier, Oriol Fornes, David J Arenillas, Chih-yu Chen, Grégoire Denay, Jessica Lee, Wenqiang Shi, Casper Shyr, Ge Tan, Rebecca Worsley-Hunt, et al. Jaspar 2016: a major expansion and update of the open-access database of transcription factor binding profiles. *Nucleic acids research*, 44(D1):D110–D115, 2016.

[72] Timothy L Bailey, Mikael Boden, Fabian A Buske, Martin Frith, Charles E Grant, Luca Clementi, Jingyuan Ren, Wilfred W Li, and William S Noble. Meme suite: tools for motif discovery and searching. *Nucleic acids research*, 37(suppl_2):W202–W208, 2009.

[73] Shobhit Gupta, John A Stamatoyannopoulos, Timothy L Bailey, and William Stafford Noble. Quantifying similarity between motifs. *Genome biology*, 8(2):R24, 2007.

[74] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[75] Todd M Lowe and Sean R Eddy. trnascan-se: a program for improved detection of transfer rna genes in genomic sequence. *Nucleic acids research*, 25(5):955, 1997.

[76] Todd M Lowe and Patricia P Chan. trnascan-se on-line: integrating search and context for analysis of transfer rna genes. *Nucleic acids research*, 44(W1):W54–W57, 2016.

[77] Dean Laslett and Bjorn Canback. Aragorn, a program to detect trna genes and tmrna genes in nucleotide sequences. *Nucleic acids research*, 32(1):11–16, 2004.

[78] Dean Laslett and Björn Canbäck. Arwen: a program to detect trna genes in metazoan mitochondrial nucleotide sequences. *Bioinformatics*, 24(2):172–175, 2007.

[79] Angelo Pavesi, Franco Conterio, Angelo Bolchi, Giorgio Dieci, and Simone Ottonello. Identification of new eukaryotic trna genes in genomic dna databases by a multistep weight matrix anaylsis of transcriptional control regions. *Nucleic acids research*, 22(7):1247–1256, 1994.

[80] Eric P. Nawrocki and Sean R. Eddy. Infernal 1.1: 100-fold faster rna homology searches. *Bioinformatics*, 29(22):2933–2935, 2013.

[81] Quan Zou, Jiasheng Guo, Ying Ju, Meihong Wu, Xiangxiang Zeng, and Zhiling Hong. Improving trnascan-se annotation results via ensemble classifiers. *Molecular informatics*, 34(11-12):761–770, 2015.

[82] Frank Jühling, Mario Mörl, Roland K Hartmann, Mathias Sprinzl, Peter F Stadler, and Joern Pütz. trnadb 2009: compilation of trna sequences and trna genes. *Nucleic acids research*, 37(suppl_1):D159–D162, 2008.

[83] Guenter Stoesser, Wendy Baker, Alexandra van den Broek, Evelyn Camon, Maria Garcia-Pastor, Carola Kanz, Tamara Kulikova, Rasko Leinonen, Quan Lin, Vincent Lombard, et al. The embl nucleotide sequence database. *Nucleic acids research*, 30(1):21–26, 2002.

[84] Weizhong Li and Adam Godzik. Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences. *Bioinformatics*, 22(13):1658–1659, 2006.