Developing computational tools to aid the design of CRISPR/Cas9 gene editing experiments

Candidate no. 1058939

Word count: 15,228 by Overleaf Word Counter

A thesis submitted for the degree of

Msc Computer Science

☐ I hereby certify that this is entirely my own work unless otherwise stated.

2022 August 29^{th}

Acknowledgements

This graduate thesis has come to an end. Here, I would like to express my heartfelt thanks to Professor Peter Minary for his supervision and guidance to me I met many difficulties at the beginning of my graduation thesis, but Professor Minary gave me a lot of encouragement and effective guidance. He made me learn to face difficulties positively and solve them. In addition, the spirit of his academic research is also worth my forever learning.

In addition, I would also like to thank Xi Xiang et al., the founder of CRISPRon, who helped me with the data for training, as well as my teacher Evangelos, my classmates and my family who gave me countless care, encouragement and help. Without their help, the completion of this graduation thesis would be difficult. They have given me many valuable opinions and suggestions in the process of completing my thesis. I would like to express my heartfelt thanks to them.

Abstract

CRISPR-Cas9, as a new generation of gene editing technology, has been widely used in biological sciences. The design of a key component of CRISPR-Cas9, the guide RNA, is important for the effectiveness and safety of the technology. In recent years, with the rise of machine learning, especially deep learning algorithms, lots of computational tools are developed to be used to assist guide RNA design. This paper aims at a difficulty in guide RNA design, which is to predict the efficiency of guide RNA. Although many tools can aid the gRNA efficiency prediction, there is still big space to improve the prediction accuracy of current tools. In this paper, a structure optimization algorithm is designed for the guide RNA efficiency prediction model and tools. The algorithm is used on an advanced existent model, CRISPRon. With my algorithm, the performance of the optimized version of the model achieves state-of-the-art performance when tested on multiple datasets.

Contents

Chapter 1

Introduction

Contents

1.1 Motivation

Gene editing technology refers to the technology of targeted modification (knockout, insertion, replacement, etc.) of genes to obtain new features or functions. As an important research field with rapid development of life science, the development and application of gene editing technology enables the genetic modification of organisms to enter an unprecedented depth and breadth, and it is also the next generation core biotechnology with the most fierce competition in the world. Early gene editing techniques include homing endonuclease (HEs) [1], zinc finger endonuclease (ZFN) [2] and transcription activator-like effector nucleases (TALENs) [3], but off-target effects or assembly complexity limit the application of these technologies in gene editing. In recent years, new gene editing technologies represented by Clustered Regularly Interspaced Short Palindromic Repeats (CRISPR)-associated protein 9 (Cas9) [4] have developed rapidly and started to be widely used in

many biological fields, including gene function research, drug development, disease treatment and crop breeding, etc.

CRISPR-Cas9 is the third generation of gene editing technology after the introduction of ZFNs, TALENs, and other gene editing technologies. The use of CRISPR-Cas as a tool for genome editing is based on the action of the Cas9-gRNA complex. Artificial guide RNA (gRNA), which is created when CRISPR RNA (crRNA) binds to transactivating crRNA (tracrRNA), is intended to match and direct the Cas protein (typically Cas9) toward specific gene targets that can be inactivated or modified depending on subsequent DNA repairing pathways. In just a few years, CRISPR-Cas9 has become popular all over the world, becoming one of the most efficient, simplest, and cheapest technologies in existing gene editing and gene modification and becoming the most mainstream gene editing system today. Selecting highly efficient gRNAs is a critical process in CRISPR gene editing tool design.

However, this technology still faces many challenges. The biggest two challenges are how to accurately predict the guide RNA (gRNA) on-target knockout efficacy and off-target profile, which would facilitate the optimized design of sgRNAs with high efficiency and specificity. This task is a tedious and extremely time-consuming process in biological experiments. The experimental design of the CRISPR-Cas system has several variables, including guide RNA design, Cas protein choice, and system delivery technique. All these factors may cause the instability of the experiment, aggravate the complexity of the experiment, and affect the success rate of the experiment. Fortunately, advances in computer science and machine learning in recent decades, especially the emergence of deep learning, have made computer-aided tools of great help for gRNA design and effect prediction.

The first step for computational tools to help gRNA design is to pick appropriate gRNAs from online gRNA libraries, which gives information on gRNAs targeting common genetic sequences. The use of computer algorithms therefore enables for an efficient high-throughput large-scale screening of the performance of the

gRNAs, considerably accelerating the design process. These tools would generally look for acceptable candidates for gRNA and predict the quality of the chosen gRNA based on its efficiency and specificity of cleavage, which require machine learning algorithms and statistical computations.

Early computer-aided tools, such as the sgRNA Scorer [5] and CHOPCHOP [6], used traditional machine learning algorithms or statistical algorithms, including Linear Regression, Support Vector Machine, Random Forest, or simple scoring algorithms that require the data to contain features related to the performance of the corresponding gRNA. The latent correlations between these features of the gRNA sequence and their efficacy (or specificity) are normally proposed by various studies. For example, the sequence GC contents, one of the features of gRNA generated, is implemented in both methods created by Doench et al. [7] and Morenno-Mateos et al. [8], where a linear regression model is trained to predict the activity of gRNAs.

The emergence of deep learning models, such as Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN), and their success in image analysis and natural language processing make researchers try to apply these algorithms to some cutting-edge computer-aided tools to help gRNA design. Compared with traditional methods that rely on manual feature extraction, deep learning can automatically extract features by data-driven. In addition, it can also learn the specific feature representation of the data set according to a large number of samples, which is more efficient and accurate in the expression of the data set, and the generalization ability of the extracted abstract features is better. DeepCRISPR, presented by Chuai et al. [9], is a landmark model which utilized the CNN to predict both on-target and off-target activity of gRNAs. Subsequently, more and more deep learning-based computational tools, such as DeepSpCas9 [10], DeepHF [11], and CRISPRon [12], are proposed, continuously improving the efficiency and specificity of gRNA design.

Although these existing tools have achieved good results in terms of on-target and off-target prediction of gRNA (i.e. efficiency and specificity) [1], there is still room for improvement with the rapid update and iteration of deep learning algorithms.

1.2 Contribution

This paper focused on improving the accuracy of predicting gRNA on-target activity (i.e., gRNA efficiency). The achievements and contributions of this paper can be summarized as follows.

1. This paper proposes a deep learning model structure optimization search algorithm specifically for predicting gRNA on-target activity.

2. The model structure obtained based on the optimization algorithm based in this paper achieved the state-of-the-art performance for predicting the on-target activity of gRNA on most data sets.

3. This paper proves that the structure of convolutional layers (especially the number of convolutional layers and size of convolutional kernels) has a great influence on the effect of CNN-based computational tools.

4. This paper summarized the most advanced models and computational tools used for predicting the on-target activity of gRNA of the CRISPR-Cas9 gene editing technology.

5. This paper provided insight for future researchers into the perspective of model architecture optimization.

1.3 Dissertation Structure Overview

This dissertation has six chapters.

[1] See more in Chapter 2

- **Introduction.** This chapter summarized the origin and development of computer-aided tools for gene editing and summarized the motivation, contribution, and general structure of this paper.
- **Background.** This chapter provides the reader with the clear and detailed background knowledge of CRISPR technology and machine learning involved in this paper.
- **Literature Review.** This chapter introduces the detailed information of several models compared in this thesis, as well as the data set the previous researchers obtained and used.
- **Methodology.** This chapter firstly analyzes the previous work and models, then proposes the model structure optimization algorithm.
- **Experiments and Results.** This chapter includes the experiments carried out, the structure of the model obtained after using the structure optimization algorithm, and the performance of the new model, Adaptive CRISPRon, compared to previous models on different data sets.
- **Discussion and Conclusion.** This chapter analyzes the results obtained before, discusses the direction of future work, and summarizes the content and work of the paper.

Chapter 2

Background

Contents

2.1 CRISPR Background

The development and iteration of CRISPR technology has brought great value to gene editing engineering and research, which is also the main biotechnology involved in this paper. This section will begin with the origin and development of CRISPR, through to the application mechanism of CRISPR-Cas9 technology in practice.

2.1.1 The Emergence of CRISPR-Cas System

Gene editing technology plays an important role in the understanding of metabolic processes in living organisms by biologists, studying gene function in living organisms and improving modern gene therapy. It plays a role in educating the biologist on the

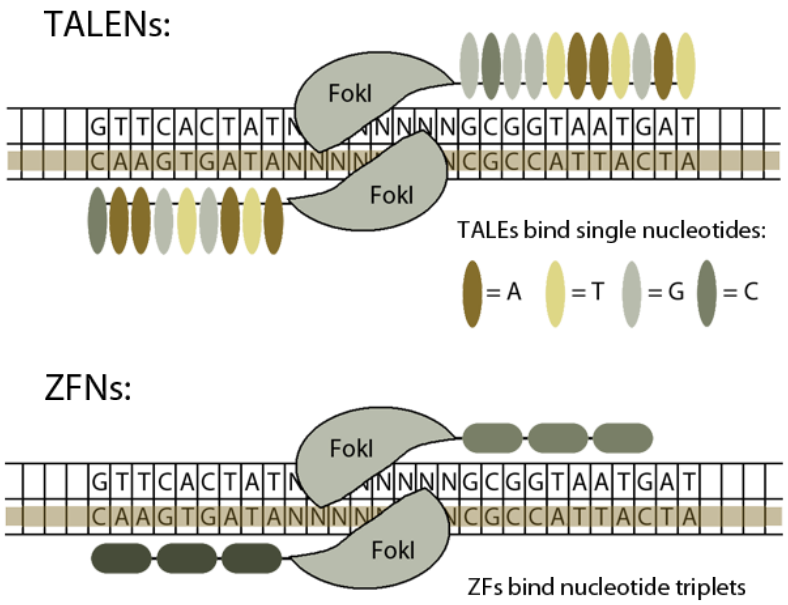

Figure 2.1: Illustration of ZFN and TALEN technologies [13]

function of genes in life. Previous genome editing technologies, such as Zinc Finger Proteins Nucleases(ZFNs) [2] and Transcription Activator-like Effectors Nucleases (TALENs) [3], have made indelible contributions to the development of genome editing. In both cases, gene editing tools are based on the interaction between proteins and DNA to position themselves on the genome, and the FokI protein [1] to splice the genome. As shown in Fig. 2.1, The anchoring of the target position by ZFNs and TALENs technique is accomplished by anchoring proteins represented by ellipses to the corresponding nucleotides (or nucleotides triplets).

However, a significant disadvantage of the anchoring process that relies on DNA-protein interaction is that the anchoring protein needs to be redesigned and synthesized when a new gene location is to be anchored, which greatly increases the workload of gene editing and makes it difficult for these technologies to adapt to high-throughput genome editing projects.

The emergence of the CRISPR (Clustered regularly interspaced short Palindromic Repeats)-Cas(CRISPR-associated) system as an innovative, powerful genome-

[1]FokI is an unusual class of enzymes that recognize a specific DNA sequence and cleave a short distance away from that sequence.

editing tool addresses the difficulty mentioned above. The CRISPR-Cas system is an acquired immune system in bacteria, which is used to fight against foreign DNA, plasmids and bacteriophages. Unlike the General immune system of ordinary prokaryotes, the CRISPR-Cas system is an acquired immune system, which means it has "memory". It can remember foreign DNA and phages that have invaded, and when they invade again, it can cut off their genomes. The severed genomes become linear, unable to be replicated and expressed, and are degraded by enzymes in the bacteria. This immune system is simple, but very useful and powerful.

Different from ZFNs and TALENs, the CRISPR-Cas system uses RNA-DNA interaction for gene localization on the genome, and only a small piece of new RNA sequence is needed for anchoring new gene positions, which greatly reduces the workload of new protein synthesis. The CRISPR-Cas system has become the new favorite of the genome editing community because it offers many advantages that other tools do not, such as ease of synthesis, ease of use, low cost, and high specificity [14].

2.1.2 Structure of the CRISPR-Cas System

The CRISPR-Cas system consists of two parts: the first part is the genes encoding Cas related proteins, as shown by the white square arrows in Fig. 2.2, which play an important role in both obtaining and splicing foreign gene fragments. The second part is called CRISPR array, which contains repeat sequence and Spacer sequence.

These two different sequences are separated, and a Spacer sequence is sandwiched between two repeat sequences. As shown in Fig. 2.2, the black diamond represents a repeat sequence and the different colored squares represent different Spacer sequences. The base composition and length of repeat sequences in the same bacteria are relatively conserved and basically unchanged. It is slightly different from one bacterium to another. Spacer sequences are used to anchor foreign genes. Because they come from different foreign genes, the base composition of Spacer sequences is quite different. The Spacer gene contains highly specific conserved

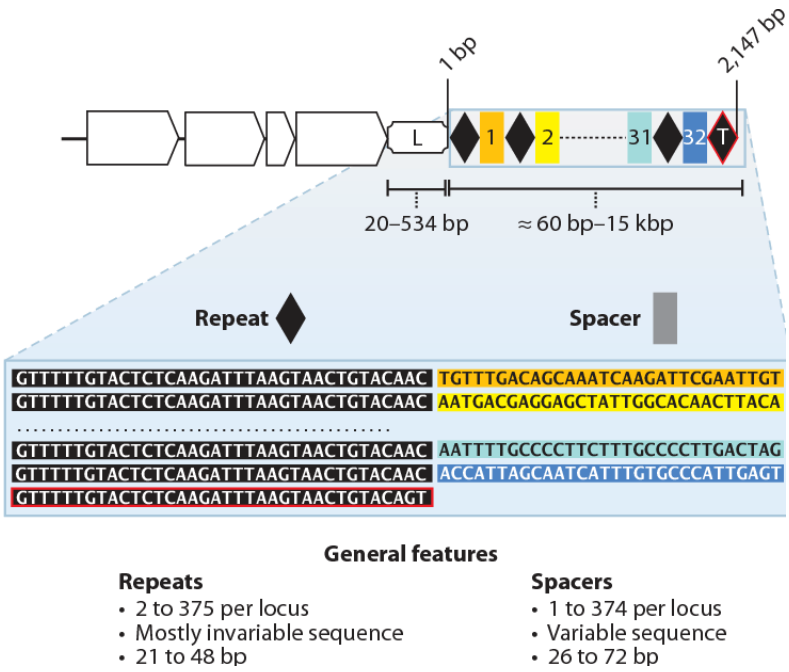

Figure 2.2: General Structure of the CRISPR-Cas System [15]

sequences in the anchored genome, ensuring that the subsequent transcribed RNA can be accurately paired with the anchored genome. CRISPR arrays are usually preceded by an Adenine-Thymine-rich leader sequence, which contains promoters and is used to initiate repeat and Spacer sequence transcription.

2.1.3 Mechanism of CRISPR-CAS System

The mechanism of CRISPR-Cas can be divided into two main parts: Adaptation and Interference.

Adaptation can be divided into two steps, namely the acquisition of Spacer sequence and CRISPR RNA (CrRNA) synthesis and processing. When a bacteriophage or a foreign gene invades a bacterium, protospacer in its genome is cleaved by Cas related genes in the CRISPR-Cas system. The identification and acquisition of Spacer by Cas protein is based on the downstream Protospacer Adjacent Motif (PAM) sequence, which plays a crucial role in Spacer acquisition and CRISPR system design in vitro. The PAM recognition sequences of different CRISPR-Cas systems are also different. When a Spacer is selected by Cas related proteins, its

gene will be cut down and inserted into the middle of the leader sequence and the adjacent repeat to form a new Spacer. In this way, the next time the same foreign gene invades, its genome can be snipped.

The second step of Adaptation is the formation of CRISPR RNA (CrRNA). As mentioned in section 3.1.4, there is a promoter in the leader sequence, which can initiate the transcription of the subsequent CRISPR array, and this transcription is continuous. So the transcribed RNA product is a long strand that contains all the spacers and repeats in the CRISPR array. This long strand is called the precursor transcript (pre-crRNA). The long pre-crRNA is then cleaved by enzymes expressed by bacterial housekeeping genes or CAS-related proteins (depending on the CRISPR system) to become mature, single-Spacer containing crRNA. The transcribed Spacer RNA sequence is complementary to the target anchor gene, and crRNA can guide Cas related proteins to shear genes in the target genome.

After the formation of a mature single Spacer crRNA, the interference step starts. The Spacer crRNA forms a complex with Cas protein and other RNA components. The crRNA can complement the gene in the foreign gene and guide the Cas protein or protein complex to splice the foreign gene fragment. The general Adaptation and Interference stages are shown in Fig. 2.3.

2.1.4 The Different CRISPR Systems

As shown in Fig. 2.4, the CRISPR-Cas system can be broadly divided into two classes, Class 1, which includes Type I, III, and IV, and Class 2, which includes Type II and Type V and Type VI. In Class 1, splicing of the foreign genome requires a protein complex consisting of more than one Cas protein and a guide RNA. In Class 2, splicing of foreign genes requires only a single splicing protein, such as the Cas9 protein in TypeII.

The Class 2 CRISPR system is most commonly used in the synthesis of Cas systems because only a single Cas9 or CpF1 protein is required for DNA splicing.

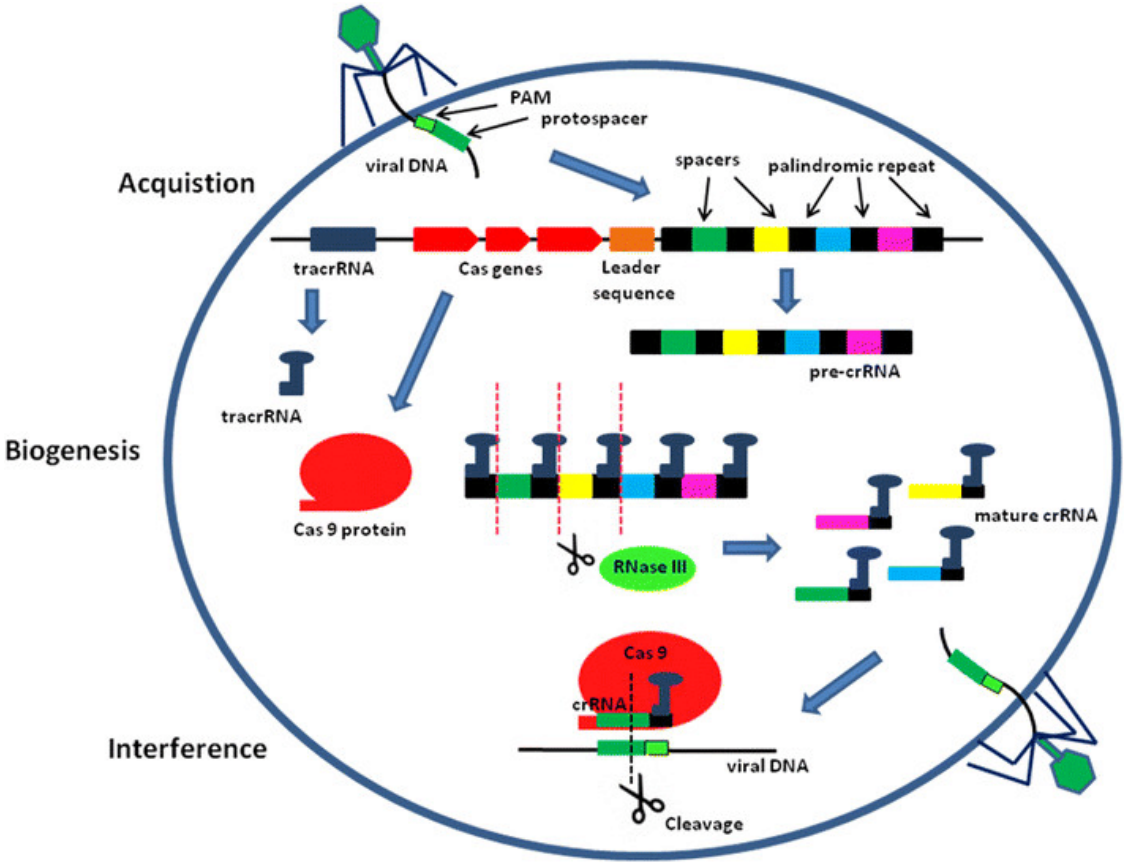

Figure 2.3: The Adaptation and Interference Stages of the CRISPR-Cas System [16]

2.1.5 The Mechanism of CRISPR-Cas9 System in Practice

The CRISPR-Cas9 mechanism is shown in Fig. 2.5. CRISPR-Cas9 belongs to the TypeII CRISPR system and is the most commonly used system. It is also the gene editing method studied in this paper. In this system, crRNA pairs complementing with noncoding trans-activating CRISPR RNA (tracrRNA) and forms a complex with Cas9 protein for DNA cleavage. This complex is called guide RNA. In practice, the guide RNA is artificially designed and is called single guide RNA (sgRNA), which combines a molecule incorporating both crRNA and trRNA.

We will also need to select genes for anchoring splicing based on PAM sequences, as described earlier. The PAM sequence of Type II system is NGG (where N is any base and G is guanine). In other words, there must be a NGG sequence after the target spliced gene fragment.

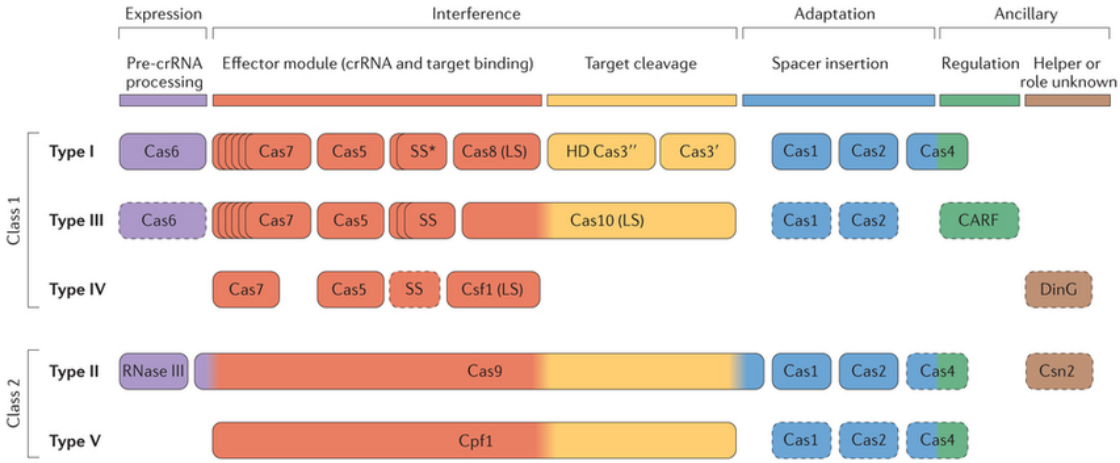

Figure 2.4: Functional Classifications of Cas Proteins [17]

After this, the sgRNA and Cas9 will be delivered into the cell to cause a cut in the designated spot. This may be accomplished by employing lentiviruses, which are viruses that insert a copy of their genome into the DNA of the cell into which they infiltrate. Encoding the production of Cas9 and sgRNA into the DNA of lentiviruses and infecting the cell with these modified viruses causes this encoding to be passed to the cell's genome, leading the cell to create both Cas9 and sgRNA. Once created, they may now connect to and cleave the specific location in the cell's DNA.

The only configurable portion of the Cas9 and gRNA complex is the crRNA, which may be altered to target different sites, since the trRNA is determined by the Cas enzyme utilized. As a result, when it comes to creating a sgRNA, the focus is on the 20-nucleotide-long crRNA. Designing guides is equal to selecting target DNA bases, since the crRNA is intended to complement the target DNA location. When a DNA double strand is cut, there are two ways to repair it. The first type is Non-homologous End Joining (NHEJ). This is the joining of broken DNA duplexes that happens spontaneously in living organisms. However, this connection method is random, which can cause base insertion, deletion, and frameshift of the reading frame. The other is Homology-directed Repair (HDR), which provides a small DNA fragment with the same sequence at both ends as the broken gene, and this

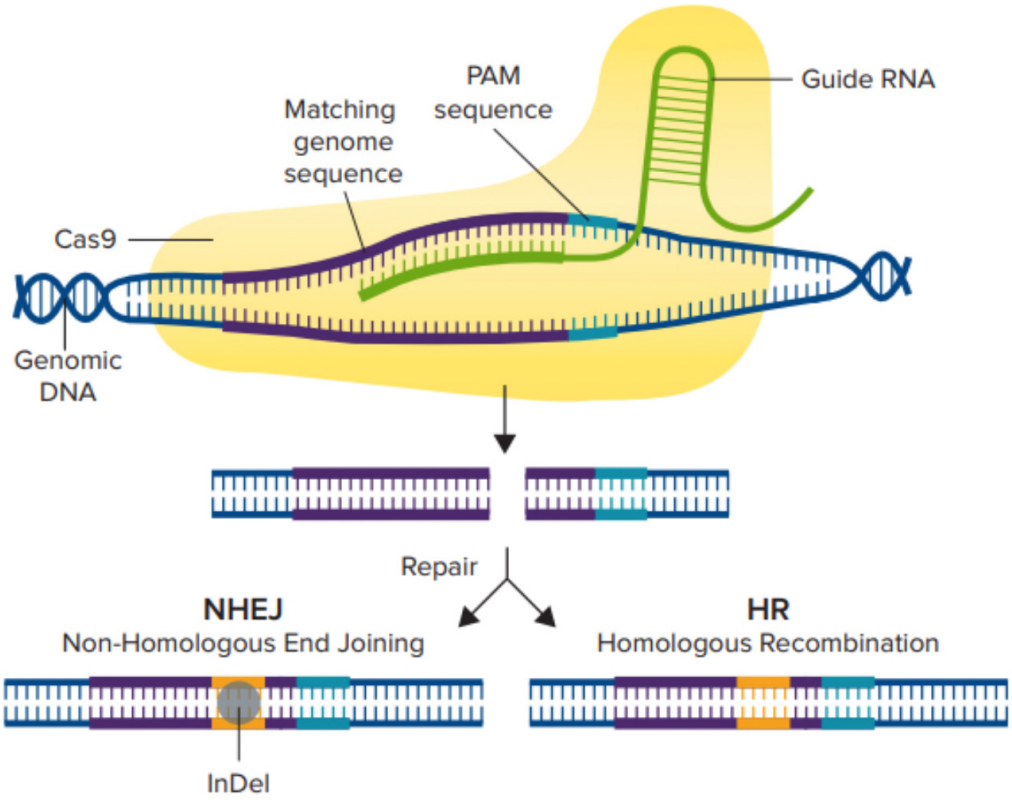

Figure 2.5: The CRISPR-Cas9 mechanism [18]

DNA fragment can be homologous recombination with the broken gene, so that genes can be inserted and deleted purposefully.

2.2 Machine Learning Background

How to improve the efficiency and specificity of the CRISPR-Cas9 system for genome editing has always been a difficult problem in this field. In recent years, machine learning has provided new ideas to solve the problems faced by CRISPR-Cas9 system. CRISPR-Cas9 system based on machine learning has gradually become a research hotspot. This section will introduce the background knowledge of the traditional machine learning models, deep learning models, and the implementation details of these models that will be discussed in later chapters in this paper.

2.2.1 Traditional Machine learning Models

2.2.1.1 Linear Regression

A linear approach to modelling the connection between a scalar answer and one or more explanatory variables is known as linear regression. Mathematically, if the data points are vectors $\mathbf{x} \in \mathbb{R}^D$, the output is $y \in \mathbb{R}$, and N observations, $\langle(\mathbf{x}_i, y_i)\rangle_{i=1}^N$ are given. A linear model assumes that y can be expressed as a linear function of the input $\mathbf{x}$.

$$y = w_0 + x_1 w_1 + \cdots + x_D w_D + \epsilon \tag{2.1}$$

where w_0 is the independent bias term and the term ϵ is noise or uncertainty [19]. Linear regression models can be fitted using least squares or by minimizing a penalized version of the least squares cost function, as in ridge regression (L2-norm penalty) [2] and lasso regression (L1-norm penalty) [3].

2.2.1.2 Random Forest

In machine learning, random forest is a classifier that contains multiple decision trees, and its output category is determined by the mode of the category output by the individual tree. Figure 2.6 shows the process of this algorithm.

[2]Ridge regression is a method of estimating the coefficients of multiple-regression models in scenarios where the independent variables are highly correlated [20].

[3]Lasso is a regression analysis method that performs both variable selection and regularization in order to enhance the prediction accuracy and interpretability of the resulting statistical model [21]

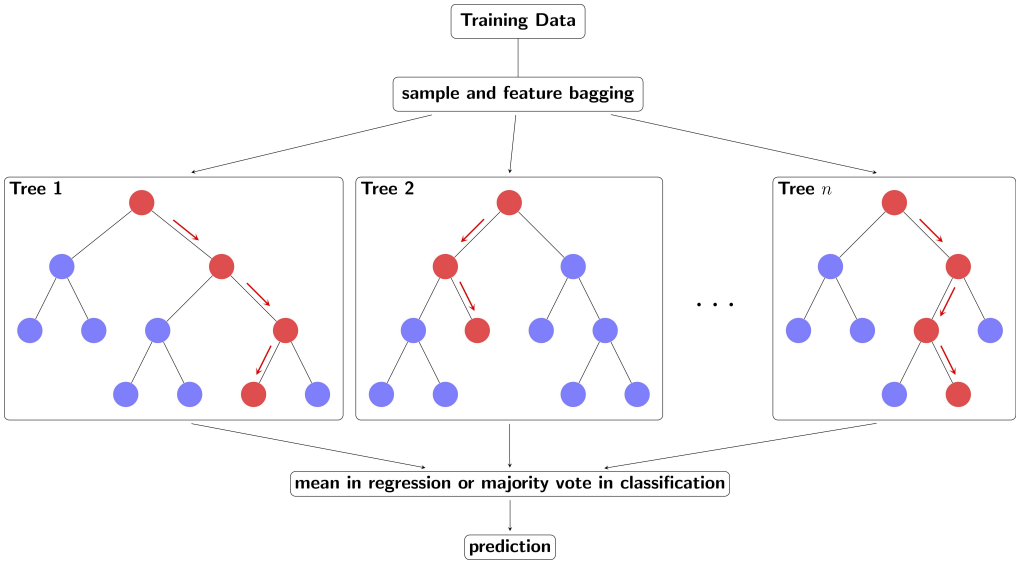

Figure 2.6: Random Forest Algorithm [22]

The first step is to partition the training data into n bags, and each of the bags will be used to train a individual decision tree. The regressor will then takes of decision of all trees into account and makes the prediction [23].

Random forest algorithm has three main hyperparameters, which need to be set before training. These include node size, the number of trees, and the number of features sampled.

One of the benefits of Random Forest is that it is not easy to overfit. When a random forest has a large number of decision trees, the classifier will not overfit the model, since averaging uncorrelated trees reduces overall variance and prediction error.

2.2.1.3 Gradient Boosting Decision Tree Regressor

The Gradient Boosting Decision Tree Regressor(GBDT) is an iterative decision tree algorithm that belongs to the boosting family [24]. Boosting is a family of algorithms that can promote a weak learner to a strong learner, which belongs to the category of Ensemble learning. Boosting is based on the idea that, for a

complex task, properly combining the judgments of multiple experts yields a better judgment than the judgments of any one expert alone.

GBDT algorithm for regression can be considered an addition model composed of M trees, and its corresponding algorithm is as follows [24].

$$F(x,w)=\sum_{m=0}^{M} a_m h_m(x,w_m)=\sum_{m=0}^{M} f_m(x,w_m) \tag{2.2}$$

where x is the input samples, w is the model parameter, h is a classified regression tree, a is the weight of each tree, and $F(x,w)$ is the prediction of the regressor.

The algorithm is as follows.

1. Initialize the weak model.

$$f_0(x)=\arg\min_c \sum_{i=1}^{N} L\left(y_i,c\right) \tag{2.3}$$

2. For $m=1...M$,

   - For each sample $i=1,2,\ldots,N$, calculate residual

   $$r_{im}=-\left[\frac{\partial L\left(y_i,f\left(x_i\right)\right)}{\partial f\left(x_i\right)}\right]_{f(x)=f_{m-1}(x)} \tag{2.4}$$

   - Use the residual of the last step as the true value of the samples, train the next tree on the data $\left(x_i,x_i m\right), i=1,2,\ldots,N$ and obtain the new tree $f_m(x)$. Its corresponding leaf node region is R_{jm}, and $j=1,2,\ldots,J$ where j is the number of leaf nodes of the regression tree.

   - Calculate the best fitting value

   $$\gamma_{jm}=\arg\min_{\gamma} \sum_{x_i\in R_{jm}} L\left(y_i,f_{m-1}\left(x_i\right)+\gamma\right) \tag{2.5}$$

   for $j=1,2,\ldots J$

   - Update the model

   $$f_m x=f_{m-1}(x)+\sum_{j=1}^{J}\gamma_{jm} I\left(x\in R_{jm}\right) \tag{2.6}$$

3. Obtain the model

$$f(x)=f_M x=f_0(x)+\sum_{m=1}^{M}\sum_{j=1}^{J}\gamma_{jm} I\left(x\in R_{jm}\right) \tag{2.7}$$

2.2.1.4 Support Vector Machine

Support Vector machines (SVM) is a binary classification model. Its basic model is a linear classifier defined in the feature space with the largest separation. The basic idea of SVM learning is to obtain a hyperplane that can correctly partition the training dataset and maximize the geometric spacing.

As shown in Figure 2.7, $\boldsymbol{w} \cdot x - b = 0$ is the separating hyperplane and $\boldsymbol{w} \cdot x - b = \pm 1$ are margins. For linearly separable data sets, there are infinitely many such hyperplanes, but the separating hyperplane with the largest geometric interval is unique.

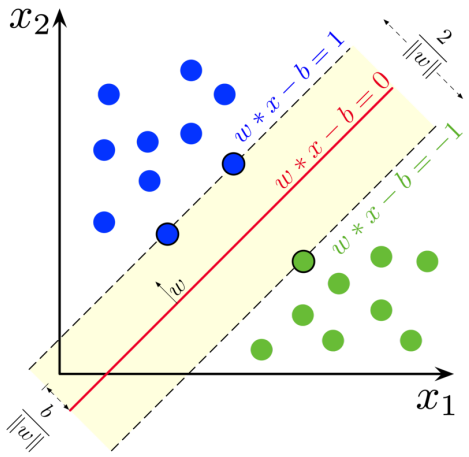

Figure 2.7: Maximum-margin Hyperplane and Margins for an SVM [25]

Mathematically, the input training dataset of SVM is $T = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$ where $\boldsymbol{x}_i \in \mathbb{R}^n$,$y_i \in \{+1, -1\}, i = 1, 2, \ldots N$, and the outputs are the separation hyperplane and the classification decision function. The algorithm is as follows.

1. Choose the hyperparameter $C > 0$, construct and solve the convex quadratic programming problem:

$$\begin{array}{ll} \min_{\alpha} & \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \left(\boldsymbol{x}_i \cdot \boldsymbol{x}_j\right) - \sum_{i=1}^{N} \alpha_i \\ \text{s.t.} & \sum_{i=1}^{N} \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C, i = 1, 2, \ldots, N \end{array} \tag{2.8}$$

and obtain the optimal solution $\boldsymbol{\alpha}^* = (\alpha_1^*, \alpha_2^*, \ldots, \alpha_N^*)^T$ where α_i is the lagrange multiplier.

2. Compute

$$\boldsymbol{w}^* = \sum_{i=1}^{N} \alpha_i^* y_i \boldsymbol{x}_i \tag{2.9}$$

Then choose α_j^* which satisfies $0 < \alpha_j^* < C$ to compute

$$b^* = y_j - \sum_{i=1}^{N} \alpha_i^* y_i \left(\boldsymbol{x}_i \cdot \boldsymbol{x}_j\right) \tag{2.10}$$

3. Compute the hyperplane $\boldsymbol{w}^* \cdot \boldsymbol{x} + b^* = 0$ and the classification decision function $f(\boldsymbol{x}) = \text{sign}\left(\boldsymbol{w}^* \cdot \boldsymbol{x} + b^*\right)$

SVM can be a non-linear classifier by using the kernel tricks. For the non-linear classification problem in the input space, it can be transformed into a linear classification problem in a certain dimensional feature space by non-linear transformation, and a linear SVM can be learned in the high-dimensional feature space. In SVM, the objective function and the classification decision function only involve the inner product between instances, therefore the inner product can be replaced with the kernel function.

Support Vector Regression (SVR) is an application of SVM to regression problem. For SVR, as long as $f(x)$ and y are close enough (between the margins and the hyperplane), the prediction is a correct prediction [26].

2.2.2 Deep Learning Models

2.2.2.1 Artificial Neural Network (ANN)

Deep learning (DL), a branch of machine learning, is an algorithm based on ANN (sometimes abbreviated as NN) for representation learning[4]. ANN, which consists of a large number of artificial neurons connected to each other, is a model that mimics the structure and function of the biological central nervous system and is

[4]It is a collection of techniques which converts raw data into a form that can be efficiently exploited by machine learning, and replaces manual feature engineering.

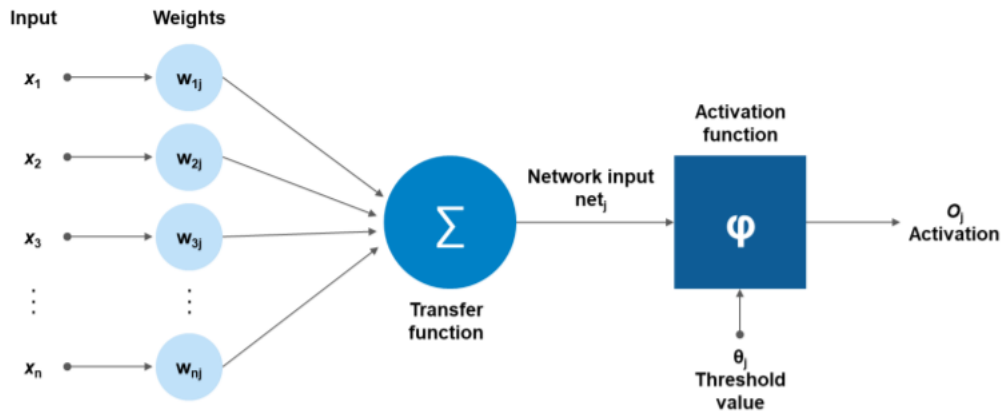

Figure 2.8: Structure of An Artificial Neuron [28]

used to estimate or approximate functions. The structure of an artificial neuron, also called a perceptron, is shown in Figure 2.8. The calculation process of a neuron is to obtain the inner product of the input vector and weight vector, and then obtain a scalar result through a nonlinear activation function. Mathematically, with input $x_1 \dots x_n$, we can calculate $o = f(\sum_1^n x_i w_i + b)$ using a perceptron, where $f(\cdot)$ is a non-linear activation function, and b is a bias term. There are many options for activation functions, the common ones are Sigmoid: $f(x) = \frac{1}{1+e^{-x}}$ and Relu: $f(x) = max(0, x)$[5].

Multiple neurons connected to each other can form a shallow neural network or a deep neural network, both are neural networks, as shown in Figure 2.9. The problem with the deep neural network was the lack of a learning algorithm, which was later proposed as the backpropagation algorithm (BP) [29]. BP is a method used in conjunction with optimization methods, such as gradient descent[6], to train ANN. It computes the gradients for all weights in the network and return them to the optimization method, which updates the weights to minimize a loss function. A

[5]Relu is often chosen as the activation function except for the output layer because it can reduce the problem of gradient vanishing and exploding [27]

[6]A first-order optimization algorithm, which, with a specified step size, iteratively searches the distance point in the opposite direction of the current point gradient on the function to find the local minima.

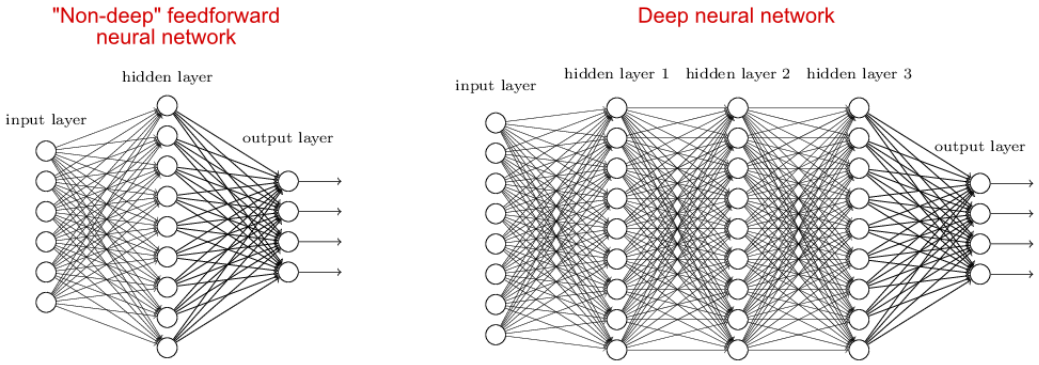

Figure 2.9: Shallow Neural Network and Deep Neural Network [30]

loss function is a mathematical function designed to compute the difference between the true output and the output of the neural network, according to the target task or the network structure. The mean square error: $\frac{1}{n}\sum_{i=1}^{n}(y_i - p_i)^2$ is often used for the regression task, and the binary cross-entropy: $-(y\log(p) + (1-y)\log(1-p))$ is often used for the binary classification, where y is the true output or label and p is the predicted output or label of the NN.

The universal approximation theorem points out the ability of deep neural networks to approximate arbitrary functions, which cannot be achieved by traditional machine learning methods.

2.2.2.2 Autoencoder

Autoencoder is a kind of neural network that uses backpropagation algorithm to make the output value equal to the input value. It first compresses the input into a latent spatial representation, and then reconstructs the output through this representation, as shown in figure 2.10.

The autoencoder consists of two parts:

- **Encoder:** This part can compress the input into a latent spatial representation, which can be represented by the encoding function $h = f(x)$.

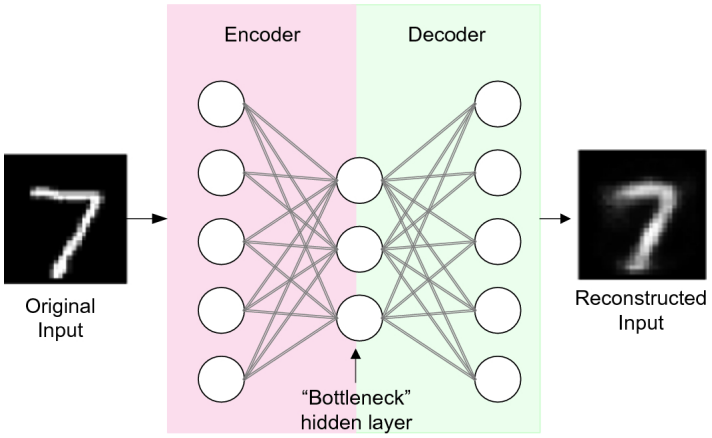

Figure 2.10: The Structure of Auto-encoder [31]

- **Decoder:** This part can reconstruct the input from the latent spatial representation, which can be represented by the decoding function $r = G(h)$.

Thus, the entire autoencoder can be described by the function $G(f(x)) = r$, where the output r is close to the original input x.

Autoencoders can learn automatically from data samples, which means that it is easy to train a specific encoder for a given class of inputs. The main applications of autoencoder are data denoising and dimensionality reduction. The trained Encoder can compress the new data and retain the most important features.

2.2.2.3 Convolutional Neural Networks (CNN)

CNN is a class of neural networks with a specific network architecture. Its artificial neurons can respond to part of the surrounding units within the coverage area and have excellent performance in large-scale image processing. Each hidden layer of CNN has two stages, convolution and pooling.

Convolution. Suppose input is a 36×36 tensors shown in figure 2.11 and a kernel is a smaller $9 \times 9 \times 1$ tensor. The kernel will take the dot product with a 9×9 patch in the image. For the first hidden layer, starting from the most left up corner $(1, 1)$ in the input, the filter slides horizontally to $(1, 27)$. Then, it moves to $(3, 1)$ and is repeated iteratively until all patches are convoluted. The

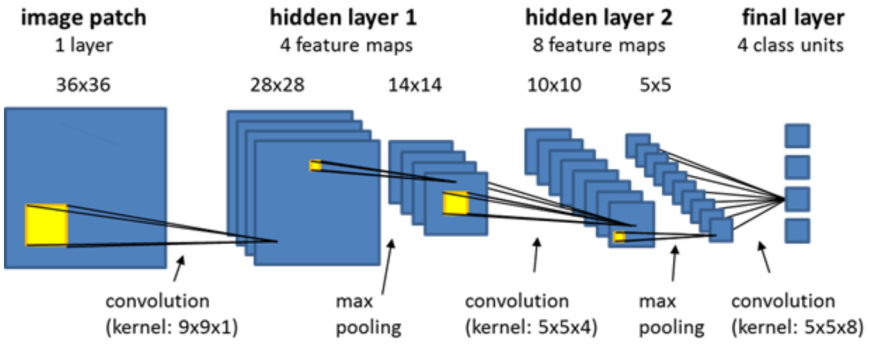

Figure 2.11: Architecture of CNN [32]

convolution operation is the same for all hidden layers. Mathematically, consider a convolutional layer between layer l and $l+1$, suppose the output of the l^{th} layer is of shape $m_l n_l F_l$ and the elements of this tensor is $a^l_{i,j,f}$, Suppose F_{l+1} filters each of shape $W_f H_f F_l$ are applied, then we can write the entries $z^{l+1}_{i',j',f'}$ as

$$z^{l+1}_{i',j',f'} = b^{l+1,f'} + \sum_{i=1}^{W_{f'}} \sum_{j=1}^{H_{f'}} \sum_{f=1}^{F_l} a^l_{i'+i-1,j'+j-1,f} w^{l+1,f'}_{i,j,f} \tag{2.11}$$

where $w^{l+1,f'}_{i,j,f}$ is the parameter for the $f^{'}th$ filter between the layer l and the index $l+1$ by (i,j,f). The gradients with respect to the parameter can be computed as :

$$\frac{\partial \ell}{\partial w^{l+1,f'}_{i,j,f}} = \sum_{i',j'} \frac{\partial \ell}{\partial z^{l+1}_{i',j',f'}} \cdot a^l_{i'+i-1,j'+j-1,f} \tag{2.12}$$

where $\frac{\partial \ell}{\partial z^{l+1}_{i',j',f'}}$ has been calculated in advance.

For backpropagation, we need compute $\frac{\partial \ell}{\partial z^l_{i,j,f}}$, which is:

$$\frac{\partial \ell}{\partial z^l_{i,j,f}} = f'\left(z^l_{i,j,f}\right) \cdot \sum_{i',j',f'} \frac{\partial \ell}{\partial z^{l+1}_{i',j',f'}} \cdot w^{l+1,f'}_{i-i'+1,j-j'+1,f} \tag{2.13}$$

where $f'\left(z^l_{i,j,f}\right)$ is the derivative of the non-linear activation function.

Pooling. Pooling operation is used to reduce the redundancy information being convoluted and to extract the most important information. There are many pooling options that can be chosen, such as max-pooling or average pooling. The max-pool

operation is the most common one, which looks at a small matrix in the input and chooses the largest value of the entries. Mathematically, suppose $\Omega\left(i', j'\right)$ is the set of all input indices that contribute to the specific patch where the pool operation is applied. The forward and backward equations for max-pooling are:

$$s_{i',j'}^{l+1} = \max_{i,j \in \Omega(i',j')} a_{i,j}^{l} \tag{2.14}$$

$$\frac{\partial s_{i',j'}^{l+1}}{\partial a_{i,j}^{l}} = 1\left((i,j) = \operatorname*{argmax}_{i,j \in \Omega(i',j')} a_{i,\tilde{j}}^{l}\right) \tag{2.15}$$

After convolution and pooling, the output will usually be aggregated through a fully connected layer. There are many variants based on CNN that can improve performance, such as VGG Net [33], Resnet [34], and Inception Net [35].

2.2.2.4 Recurrent Neural Networks (RNN)

DNN and CNN are both Feedforward Neural Networks, since there are no 'loops' in their architectures. However, RNN is a kind of neural network where the hidden layer is a function of both the input x and the previous hidden layer h_{t-1} as shown in Figure 2.12, where t is the time step. It is often used for temporal data and time series forecasting.

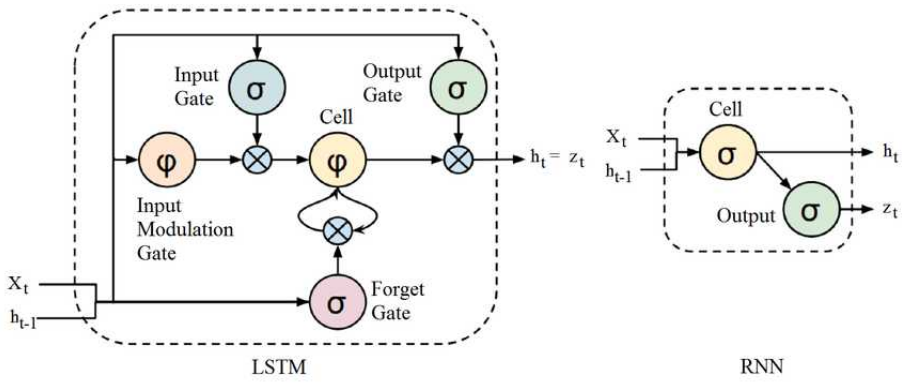

Figure 2.12: LSTM and RNN Unit Architecture [30]

Suppose w_o and w_h are the weights of the hidden layer and the output layer, respectively. The hidden states and outputs at each time step are:

$$\begin{aligned} h_t &= f\left(x_t, h_{t-1}, w_h\right) \\ o_t &= g\left(h_t, w_o\right) \end{aligned} \tag{2.16}$$

where f and g are activation functions of hidden layer and the output layer.

The application of backprogration to unrolled recurrent networks is called the Backpropagation Through Time (BPTT) algorithm. The gradients of the with respect to the parameters w_h of the objective function L can be calculated using the following equations:

$$\begin{aligned} \frac{\partial L}{\partial w_h} &= \frac{1}{T}\sum_{t=1}^{T}\frac{\partial l\left(y_t, o_t\right)}{\partial w_h} \\ &= \frac{1}{T}\sum_{t=1}^{T}\frac{\partial l\left(y_t, o_t\right)}{\partial o_t}\frac{\partial g\left(h_t, w_o\right)}{\partial h_t}\frac{\partial h_t}{\partial w_h} \end{aligned} \tag{2.17}$$

where T is the total time steps, l is the lost with regard to output o_t and true label y_t, and $\frac{\partial h_t}{\partial w_h}$ can be computed by :

$$\frac{\partial h_t}{\partial w_h} = \frac{\partial f\left(x_t, h_{t-1}, w_h\right)}{\partial w_h} + \sum_{i=1}^{t-1}\left(\prod_{j=i+1}^{t}\frac{\partial f\left(x_j, h_{j-1}, w_h\right)}{\partial h_{j-1}}\right)\frac{\partial f\left(x_i, h_{i-1}, w_h\right)}{\partial w_h} \tag{2.18}$$

Recursive Neural Networks are a generalization of RNN. It is structured in a tree-like form and can model hierarchical structures in input.

RNN has difficulty learning long-term dependencies and may have gradient vanishing or gradient exploding problem. Therefore, a special kind of RNN, Long Short Term Memory Network(LSTM), is introduced by Hochreiter and Schmidhuber [36]. LSTM has three gates:

1. forgot gate: $f_t = \sigma\left(W_f \cdot \left[h_{t-1}, x_t\right] + b_f\right)$

2. input gate: $i_t = \sigma\left(W_i \cdot \left[h_{t-1}, x_t\right] + b_i\right)$

3. output gate: $o_t = \sigma\left(W_o \cdot \left[h_{t-1}, x_t\right] + b_0\right)$

Therefore, on the current time step, the state of unit is $c_t = f_t \circ c_{t-1} + i_t \circ \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$ and the output of layer is $h_t = o_t \circ \tanh(c_t)$. The unit structure of LSTM allows gradient to be transmitted well, which greatly reduces the problem of gradient vanishing and exploding.

2.2.3 Implementation Details of Models

2.2.3.1 Metrics

Trained machine learning models need metrics to measure how well they perform on test sets. Metrics for measuring model performance vary for different tasks. This subsection will discuss several metrics used to measure the performance of models that will later be mentioned in this paper.

1. **Mean Square Error(MSE).**

   One of the common metrics used for regression tasks is MSE. It is a measure reflecting the degree of difference between the estimated value and the true value. It gives an absolute number on how much the predicted results deviate from the actual number.

   Mathematically,

   $$MSE = \frac{1}{n}\sum_{t=1}^{n}(y_{true} - y_{pred})^2 \tag{2.19}$$

   where:

   n = number of data points

   y_{pred} = the prediction value of model

   y_{true} = the actual value

2. **Mean Absolute Error(MAE)**

   MAE is another common metric that can be used for regression tasks.

   Mathematically

   $$MAE = \frac{1}{n}\sum_{t=1}^{n}|y_{true} - y_{pred}| \tag{2.20}$$

where:

n = number of data points

y_{pred} = the prediction value of model

y_{true} = the actual value

3. **Pearson's Correlation Coefficient**

Pearson's correlation coefficient, usually denoted by the letter r or ρ, measures the linear relationship (or degree of linear correlation) between two random variables. The Pearson correlation coefficient of the population between two variables is defined as the quotient of the product of the covariance and standard deviation between two variables, as follows:

$$\rho_{X,Y} = \frac{\mathbb{E}\left[(X - \mu_X)(Y - \mu_Y)\right]}{\sigma_X \sigma_Y} \tag{2.21}$$

The sample Pearson correlation coefficient is defined as follows:

$$r = \frac{\sum_{i=1}^{n}\left(x_i - \bar{X}\right)\left(y_i - \bar{Y}\right)}{\sqrt{\sum_{i=1}^{n}\left(x_i - \bar{X}\right)^2}\sqrt{\sum_{i=1}^{n}\left(x_i - \bar{Y}\right)^2}} \tag{2.22}$$

where:

σ_X and σ_Y are the the standard deviation of X and Y, respectively

μ_X and μ_Y are the mean of X and Y, respectively

$\bar{X}$ and $\bar{Y}$ are the sample mean of X and Y, respectively

$\mathbb{E}$ is the expectation

Pearson's coefficient equals 1 indicates perfect positive linear correlation between two variables, equal to -1 indicates perfect negative linear correlation between variables, and equal to 0 indicates no linear correlation between variables.

4. **Spearman's Coefficient of Correlation**

   Spearman's correlation can be regarded as the non-parametric version of Pearson's correlation. Pearson's correlation is a statistical measure of the strength of the linear relationship between two random variables, while Spearman's correlation examines the strength of the monotonic relationship between them. Pearson's correlation coefficient is calculated using the data sample value itself, while Spearman's correlation coefficient is calculated using the data sample rank value.

   For a sample of size n, the n raw scores X_i, Y_i are converted to ranks $\mathrm{R}(X_i), \mathrm{R}(Y_i)$, and r_s is computed as

$$r_s = \rho_{\mathrm{R}(X),\mathrm{R}(Y)} = \frac{\mathrm{cov}(\mathrm{R}(X), \mathrm{R}(Y))}{\sigma_{\mathrm{R}(X)}\sigma_{\mathrm{R}(Y)}}, \tag{2.23}$$

   where:

   ρ denotes the usual Pearson correlation coefficient, but applied to the rank variables

   $\mathrm{cov}(\mathrm{R}(X), \mathrm{R}(Y))$ is the covariance of the rank variables

   $\sigma_{\mathrm{R}(X)}$ and $\sigma_{\mathrm{R}(Y)}$ are the standard deviations of the rank variables.

   Spearman's correlation coefficient is suitable when the data have nonlinear relationships, when one side of the data is ordinal, or when there are significant outliers in the data.

5. **Classification Metrics:**

   Accuracy, precision, recall, F1 score, and Area Under Receiver Operating Characteristic Curve (AUROC) are commonly used to evaluate classification tasks. Accuracy is the proportion of the total number of predictions that were correct. Precision is the proportion of positive cases that were correctly identified. Recall is the proportion of actual positive cases which are correctly identified. F1 score $= \frac{2}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}}$. The ROC curve is the plot between

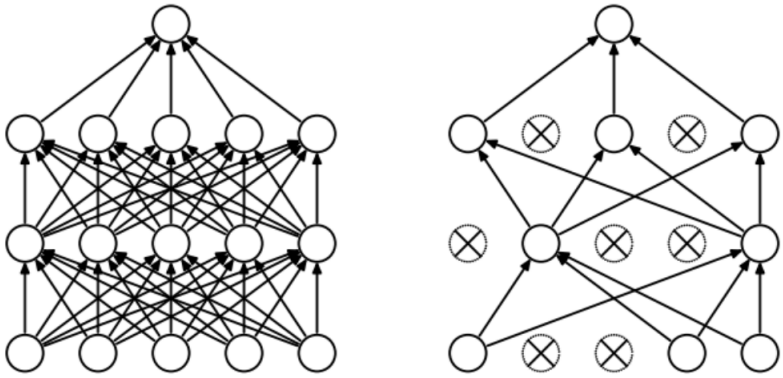

Figure 2.13: Dropout Neural Net Model [39]

true positive rate and false positive rate of the model, and AUROC is just the area under the ROC curve.

2.2.3.2 Performance Improvement

A competent deep learning model may reliably anticipate results while requiring little training time. Preventing overfitting with regularization strategies such as dropout [37] and weight decay [38] is a key strategy to enhance model accuracy. Dropout means random dropping of some units for each layer of the network, as shown in figure 2.13. Since the dropout strategy randomly drops some units at each layer, the trained network is relatively small. Moreover, since every feature is likely to be dropped, the whole network will not be biased to a certain feature, which can alleviate overfitting. weight decay is another way to prevent overfitting. In the loss function, weight decay is a coefficient placed in front of the regularization term. The regularization term generally indicates the complexity of the model, so the effect of weight decay is to regulate the impact of model complexity on the loss function. If weight decay is large, then the value of the loss function of the complex model is also large.

To reduce the training time cost, the model convergence speed can be accelerated by using techniques such as batch normalization [40] or Adaptive Moment Estimation

(Adam) optimizer [41]. Batch normalization is a technique for making neural network training quicker and more stable by normalizing the inputs of the layers by re-centering and re-scaling. It was thought to alleviate the problem of internal covariate shift, in which parameter initialization and changes in the distribution of each layer's inputs alter the network's learning rate. Some researchers have recently claimed that batch normalizing enhances performance. [25, 42, 43] Adam is a more advanced function optimization method than the traditional Stochastic Gradient Descent or Min-batch Gradient Descent. It can make the function fit the high dimensional data quickly.

2.2.3.3 Hyperparameters Searching Algorithms

Hyperparameters are parameters that are used to control the behavior of an algorithm when building a model. These parameters cannot be learned from the normal training process. They need to be assigned before training the model.

One of the hardest parts of a machine learning workflow is finding the best hyperparameters for the model. The performance of machine learning models is directly related to hyperparameters. Hyperparameter tuning can be performed in the following four methods:

- **Manual Tuning:**

  The random hyperparameter set is checked manually to train the model, and the most appropriate parameter set is selected. This time-consuming method requires a lot of experimentation and does not guarantee the best combination of parameters.

- **Grid Search:**

  Grid search is similar to manual tuning. It builds a model for each permutation of all given hyperparameter values specified in the grid and evaluates and

selects the best model. However, since it tries every combination of hyperparameters and chooses the best combination based on the cross-validation score, it is extremely time-consuming.

- **Random Search:**

  The motivation for using a random search instead of a grid search is that in many cases, all hyperparameters may not be equally important. Random search randomly selects a combination of parameters from the hyperparameter space, and the parameters are selected according to a given number of iterations. Random search is faster than grid search, but its problem is that it is not guaranteed to give the best combination of parameters.

- **Bayesian Optimization:**

  Bayesian optimization belongs to a class of optimization algorithms called Sequential Model-based optimization (SMBO) [44]. These algorithms use previous observations of losses to determine the next best point to sample. Its general steps are as follows:

  1. According to the existing tuning history $\mathcal{H} = (x_{1:k}, f(x_{1:k}))$, where $\mathcal{H}$ is the history, the record of all previous records of $\{x, f(x)\}$, the probability distribution model M is established.
  2. Acquisition function is used to select the next hyperparameter x_{k+1};
  3. Add the new observation $(x_{k+1}, f(x_{k+1}))$ to $\mathcal{H}$.
  4. Repeat steps 1-3 until the maximum number of iterations is reached

  Therefore, the main difference between different Bayesian optimization methods is which probabilistic model is used to model the history and how to choose the acquisition function.

The Tree-structured Parzen Estimator (TPE) algorithm [45] that will be involved later in this paper is a kind of Bayesian optimization algorithm. As mentioned earlier, two of the key steps in Bayesian optimization are:

- To model the objective function to be optimized, obtain the distribution of the function $p(y \mid x)$ (x is the hyperparameter value, y is the objective function), and thus understand the range in which the function is likely to fluctuate.

- To design an acquisition function to help us make the judgment. Expected Improvement (EI) is the most common design plan, which deals with a compromise between Exploration [7] and Exploitation [8]. The formula of EI is:

$$EI_{y^*}(x) = \int_{-\infty}^{+\infty} \max\left(y^* - y, 0\right) p_M(y \mid x) dy \tag{2.24}$$

  where y^* is a threshold, EI is the expectation that measures how much y is improved on average relative to the threshold y^*. Therefore, the next hyperparameter we are looking for is:

$$x_{new} = \operatorname{argmax} x_x EI_{y^*}(x) \tag{2.25}$$

  The formula of EI reflects the exploration and exploitation trade-off [46].

TPE adopts a special way of thinking to model probability distributions. According to Bayes' theorem: $p(y \mid x) = \frac{p(x|y)p(y)}{p(x)}$. TPE dividing $p(y \mid x)$ into $p(x \mid y)$ and $p(y)$, and the $p(x \mid y)$ is divided into:

$$p(x \mid y) = \begin{cases} l(x), & y < y^* \\ g(x), & y > y^* \end{cases} \tag{2.26}$$

As shown in equation 2.26, TPE constructs different distributions for the observation points x on both sides of the threshold y^*, which can be considered as hyperparameter probability distributions for good grades and for bad grades. The hyperparameter

[7] The likelihood of a algorithm to search the unexplored space
[8] The likelihood of a algorithm to search around the found optimal value.

γ is used to select the threshold, which is the quantile of y^*, and therefore $p\left(y < y^*\right) = \gamma$. The division gives:

$$\begin{aligned} p(x) &= \int_R p(x \mid y)p(y)dy \\ &= \int_{-\infty}^{y^*} p(x \mid y)p(y)dy + \int_{y^*}^{+\infty} p(x \mid y)p(y)dy \\ &= \gamma l(x) + (1-\gamma)g(x) \end{aligned} \tag{2.27}$$

and

$$\begin{aligned} EI_{y^*}(x) &= \int_{-\infty}^{+\infty} \max\left(y^* - y, 0\right) p_M(y \mid x)dy \\ &= \int_{-\infty}^{y^*} \left(y^* - y\right) p(y \mid x)dy \\ &= \int_{-\infty}^{y^*} \left(y^* - y\right) \frac{p(x \mid y)p(y)}{p(x)}dy \\ &= \int_{-\infty}^{y^*} \left(y^* - y\right) \frac{l(x)p(y)}{\gamma l(x) + (1-\gamma)g(x)}dy \\ &= \frac{\int_{-\infty}^{y^*} \left(y^* - y\right) p(y)dy}{\gamma + (1-\gamma)\frac{g(x)}{l(x)}} \end{aligned} \tag{2.28}$$

From the above equations, $EI_{y^*}(x) \propto \left(\gamma + (1-\gamma)\frac{g(x)}{l(x)}\right)^{-1}$. When γ is certain, the denominator depends only on the ratio of $\frac{l(x)}{g(x)}$, and the x the algorithm looking for is the x that maximize $\frac{l(x)}{g(x)}$.

Chapter 3

Literature Review

Contents

On-target activity (efficiency) prediction of sgRNA is an important task in the CRIPSR-Cas9 gene editing. Before this paper, there have been a lot of computational models and tools that can predict on-target activity of sgRNAs, so as to assist the design and screening of sgRNAs. In this chapter, I will first introduce computational models and tools with leading effects. Then, I will introduce the CRISPRon model proposed by Xi et al. [12] and data they generated in detail, since CRISPRon is the mainly referenced work for my model in this paper.

3.1 The Tools and Datasets Before CRISPRon

3.1.1 Tools Based on Traditional Machine Learning Algorithms or Biological Features

Before the current optimal CRISPRon model was proposed, many computational tools have been available to assist the design of sgRNA, including those based on traditional machine learning algorithms (or biological indicators) and those based on deep learning models.

Computing tools based on traditional machine learning algorithms or biological indicators are mainly as follows:

- **CHOPCHOP [6]:** a tool to score sgRNA efficiency based on several scoring indicators, including G20 [1], as well as the scoring methods proposed by Xu et al. [47], Doench et al. [7], and Moreno-Mateos et al. [8].

- **Wu-CRISPR [48]:** a tool to score sgRNA efficiency that identified 310 characteristics, including the classic characteristics in bioinformatics such as GC content [2], alignment results [3], purine and pyrimidine content, etc., that are characteristic of highly potent sgRNAs.

- **Tuscan [49]:** a tool to score sgRNA efficiency built by random forest algorithm considering features based on counts and positions for nucleotides and dinucleotides.

- **PhytoCRISP-Ex [50]:** a tool that accepts or rejects a sgRNA base on the guanine position and off-target activity.

- **FlashFry [51]:** A tool that considers the Doench et al. [7], and Moreno-Mateos et al. [8] scoring methods as well as some other features such as GC-content

[1] the metric that accepts sgRNA with guanine at the 20^{th} position
[2] the percentage of nitrogenous bases that are guanine (G) or cytosine (C)
[3] a way of arranging sequences to identify regions of similarity

- **sgRNA Scorer 2.0 [5]:** A tool to score the efficiency of sgRNA based on the support vector machine.
- **SSC [47]:** A tool to score sgRNA efficiency that based on 28 sequence features that are highly related to the efficiency.
- **Azimuth:** [**52**] One of the state-of-the-art predictive approaches to modeling which RNA guides will effectively perform a gene knockout by way of the CRISPR/Cas9 system. The team firstly identified features that are important for prediction such as nucleotide identity, which are helpful such as thermodynamics, and which are redundant such as microhomology; then they combined their insights of useful features with exploration of different model classes, settling on one model which performs best, which is the GBDT. Azimuth project is sponsored by Microsoft and cited Donech et al.'s work in 2016 [53] as reference for the useful features.

These tools do not use deep learning algorithms, and most of the tools are based on biological features related to the efficiency of sgRNA derived from experimental data and traditional machine learning algorithms. The disadvantages of these models are that the features used for modeling may not be comprehensive and accurate enough, and the complexity of the algorithms used for modeling is too simple. However, modeling by a deep learning algorithm can avoid manual selection of model features. Moreover, due to the universal approximation theorem mentioned in Chapter 2, most deep learning models have higher model complexity and expressive ability and will have better performance when the data are sufficient.

3.1.2 DeepCRISPR

DeepCRISPR [9] is a comprehensive computational tool that uses deep learning to predict sgRNA on-target and off-target activity. It completely automates the identification of sequences and characteristics that may impact the efficiency of sgRNA. As shown in figure 3.1, to predict activity on target, DeepCRISPR will

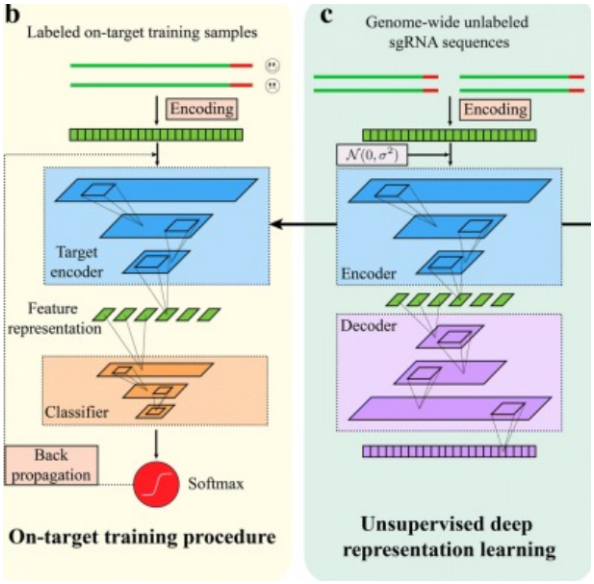

Figure 3.1: DeepCRISPR for sgRNA On-target Activity Prediction [9]

first take unlabeled sgRNA sequences as input to train an autoencoder. It will then use the trained encoder to compress the labeled on-target training sequences to get the critical feature representations. Finally, it will use these feature representations of the training inputs to train a CNN classifier with a backpropagation algorithm, which can be used to classify whether an sgRNA is efficient. DeepCRISPR can complete both the classification task and the regression task. It has a variant on GitHub [4] that can be used to score a given sgRNA.

3.1.3 DeepHF

Wang et al. measured gRNA activity for two highly specific SpCas9 variants [5] and wild-type SpCas9 in human cells, and generated new data sets [11]. They also measure and pick the most important features for sgRNA efficiency. Their generated data sets are used to train their model, and the features they obtained are later concatenated in their DeepHF model structure.

[4]Please check `https://github.com/bm2-lab/DeepCRISPR`

[5]SpCas9 is the S. pyogenes Cas9, which can be used interchangeably as Cas9. SpCas9 variants are often used instead of SpCas9, as there may be problems such as off-target effects or PAM sequence lack that SpCas9 cannot handle

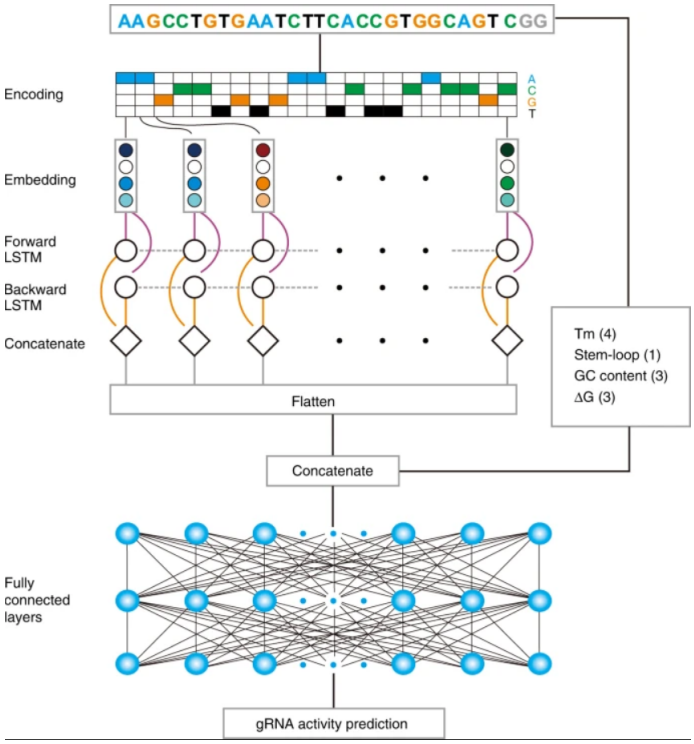

Figure 3.2: DeepHF architecture for the prediction of sgRNA on-target activity [11]

The architecture of their model is shown in figure 3.2. Their model is to handle the regression problem and the output is the indel frequency [6] in range [0,1]. With a new input sgRNA sequence, their model first used the one-hot encoding method to encode the input sequences. Then it projected the input matrix $\mathbf{x} \in \mathbb{R}^{L\times 4}$ to the dense real-valued space $\mathbf{E} \in \mathbb{R}^{L\times m}$ by the embedding weight matrix $\mathbf{W}_m$, following a Bidirectional long short-term memory neural network (BiLSTM). The BiLSTM can take advantage of LSTM (as mentioned in Chapter 2) and processes input data both in the forward and backward orders, allowing to combine the gRNA sequence information from both directions in every time step. The output of the BiLSTM will then be flattened, concatenated with the critical features they found and passed through a fully connected network to get the efficiency prediction.

[6] A metric for efficiency which will be introduced in detail later in next section

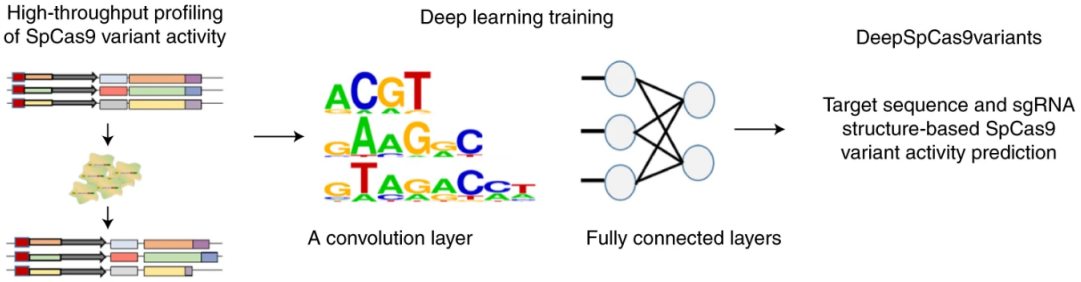

Figure 3.3: General structure of DeepSpCas9 variants for the prediction of sgRNA on-target activity [54]

3.1.4 DeepSpCas9

DeepSpCas9 [10] is another model that can be used to predict SpCas9 activity on target. DeepSpCas9 is based on a CNN architecture that has one convolutional layer and one pooling layer at the front, as well as three fully connected layers with a dropout rate of 0.3 in each layer. The adopted convolutional layer includes an inception module with a total of 210 filters (100, 70, and 40 filters at 3, 5, and 7nt in length, respectively). The pooling layer and three fully connected layers use ReLU activation functions. This architecture was later referenced and used in the design of the CRISPRon model.

3.1.5 DeepSpCas9variants

DeepSpCas9variants [54] is very similar to DeepSpCas9 except it is used to predict the on-target activity of sgRNA with SpCas9 variants. Kim et al. compared the efficiency and specificity of thousands of sequences and trained their model on the data set they generated. The model structure is similar to DeepSpCas9 except that they only used one fully connected layer at the end of the model. The general structure of DeepSpCas9variants is shown in figure 3.3.

3.2 CRISPRon and Xi et al.'s Dataset

One serious problem with pre-Crispron models is that because they are trained and tested on different data sets, their performance on new data can be significantly affected. However, due to the different experimental procedures and standards, these independent datasets have poor compatibility with each other, so they cannot be used to train the model uniformly. In order to create more precise prediction algorithms, it is crucial to produce more data from gRNA activity that is consistent with earlier research.

In 2021, Xi Xiang et al. generated on-target gRNA activity data for 10,592 SpCas9 gRNAs [12]. Then they integrated them with the dataset generated by Kim et al. [10, 54] and obtained 23,902 gRNAs. They used these data to train a deep learning model, CRISPRon. CRISPRon performs the best Spearman's correlation score at multiple datasets on on-target sgRNA activity score predictions with its special model structure and the high-quality data trained it. In this section, I will introduce the details of CRISPRon and the dataset they generated.

3.2.1 Data Generation

Xi et al. developed a high-throughput technique for measuring sgRNA activity in cells to obtain high-quality CRISPR on-target gRNA activity data. Xi Xiang et al. created a pool of 12,000 sgRNA oligos targeting 3834 human protein-coding genes and transduced the sgRNA library into SpCas9-expressing and wild-type HEK293T cells. The pipeline was then built to analyze the CRISPR-induced indels. Since SpCas9 will cause insertions and deletions, the length of the edited sequence is expected to change from its original 37 base pairs (bp) [7]. Therefore, the equation of indel frequency is defined as:

$$\text{Indel Frequency} = \frac{(\text{ Number reads with length } \neq 37\text{bp})}{(\text{Total number of reads})}\% \tag{3.1}$$

[7]The length of the sequences used to be edited by the sgRNA in the experiments.

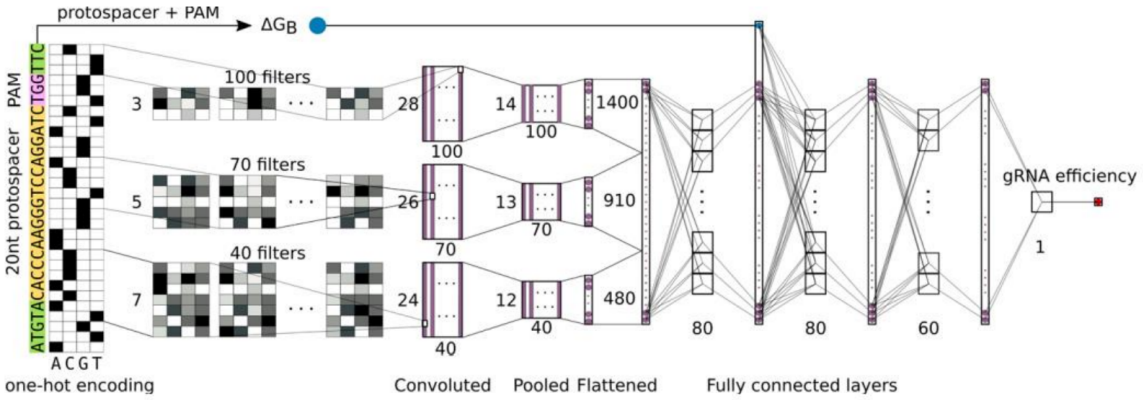

Figure 3.4: Architecture of CRISPRon [12]

Targeted deep sequencing was used to examine the frequency of indels in cells 2, 8, and 10 days after transduction. The frequency of indel (on-target activity) of gRNAs on days 8 and 10 was highly associated. For future investigations and model development, the average gRNA activity of days 8 and 10 was used. As a consequence, Xi et al. were able to gather high-quality gRNA activity data for 10,592 gRNAs, which is a great resource to improve the quality of CRISPR-gRNA designs.

3.2.2 Model Architecture and Training Process

Xi Xiang et al. employed a one-hot encoding of the input sequence with length 30 [8] which was fed into a number of 3, 5, and 7-sized filters acting directly on the one-hot encoded sequence. The convolutions, which are the outputs of the filters, were flattened and pass to two fully connected layers before output the gRNA on-target activity was output, as shown in figure 3.4.

The number of weights and arrangement of the convolutions, as well as those of the two final fully linked layers, are the same as in DeepSpCas9. [10]. They did not further tune the hyperparameters. However, trained a gradient boosting regression tree (GBRT) [9] model and applied two methods, the Shapley Additive

[8]The inputs are the **complementary DNA sequence** to 20 nucleotides (nt) sgRNA, 3nt PAM and 7nt context

[9]It is identical to GBDT mentioned in Chapter 2 except it is used for regression task

exPlanations (SHAP) [10] and the Gini importance [11], for feature analysis. Through several features including GC content and folding energy of the spacer gRNA, they found the gRNA-target-DNA binding energy, ΔG_B [57], is a far better biological feature of on-target activity for sgRNA than others. Concatenation of ΔG_B with the output of the convolutional layers improved the MSE from 144.73 to 140.83. The model in figure 3.4 thus became their final CRISPRon model.

Using 6-fold cross-validation, CRISPRon was trained on the combined dataset with 23,902 gRNAs that had been divided into six parts. After 10 iterations, the top models from each of the six validation sets are averaged to obtain the final CRISPRon result.

For comparison, they also trained other three models, pre-CRISPRon-v0, pre-CRISPRon-v1, and CRISPR-v0 with different utilization of their dataset, where:

- **pre-CRISPRon-v0** is trained on the dataset generated by themselves (not the combined one with Kim et al.'s) with a 5-fold cross-validation while using the 6^{th} partition as an internal independent test set.
- **pre-CRISPRon-v1** is trained on the dataset generated by themselves as well, with all six partitions with 6-fold cross-validation.
- **CRISPRon-v0** is trained on the combined data set with a 5-fold cross-validation (not the 6-fold cross-validation as in CRISPRon) to evaluate the model on 6^{th} the internal independent test set.

[10] SHAP seeks to explain the prediction made for that data point by calculating how much each feature contributed to the prediction produced for a given input datapoint[55]

[11] an index that determines the likelihood that a certain feature would be erroneously categorized when chosen at random. [56]

Chapter 4

Methodology

Contents

4.1 Methodology Analysis of Previous Work

By generating high-quality training data, combining data from predecessors, and meticulous screening of biological features, Xi et al. enabled their trained deep learning model CRISPRon to achieve state-of-the-art performance. However, they did not spend efforts on optimizing the model architecture, which is critical to biological sequence analysis.

One of the advantages of deep learning models over traditional machine learning models is that the deep learning models can automatically extract representations from the inputs. With different model architectures, the extracted representations in the hidden layers of deep learning models can vary on a large scale. Therefore, different architectures of models can lead to huge gaps in model performance for a particular type of input. As shown in figure 3.4, in the first convolutional layer, CRISPRon used many convolutional 1D filters with different kernel sizes to extract the features from the one-hot embedding of the input sequence. Unlike the 2D or 3D convolution filters mentioned in Chapter 2, the 1D convolution filter used in

CRISPRon is rectangular and only slides in a one-dimensional direction to extract features. In each calculation, these convolution filters will take the dot product of the one-hot encoding information of a certain number of nucleotides with their weights according to their different convolution kernel sizes. This means that filters with different nuclear sizes extract information from different nucleotide lengths per calculation. Biologically, certain nucleotide lengths may contain important features. For example, codons are triplet sequences of nucleotide residues on mRNA (or DNA) that encode a specific amino acid. For such sequences, if only filters whose convolution kernel size is not three are used to complete the convolution operation, they may not be able to extract the important features in the sequence well, so that the weights in the hidden layer may deviate from the optimal weights (or it is difficult to converge to the optimal weights in the training process), and finally affect the performance of the model.

As mentioned in Chapter 3, the CRISPRon model structure adopts the model structure of DeepSpCas9 [10] without further optimization. Although for DeepSpCas9, Kim et al. interrogated the hyperparameters of structures, they only interrogated a very small hyperparameter space. Their search procedure was as follows:

1. They first decided to use three separated 1D convolution filter types (each type has a different kernel size and has many filters) at the first layer as shown in figure 3.4. This number is fixed, which means that they did not search for any space of convolutional filter with different kernel sizes other than three.

2. Then they used a random search algorithm to search through the structural hyperparameter space, where the first convolutional filter kernel size is from range 2 to 4, the second filter kernel from range 5 to 6, and the third filter kernel is from range 7 to 9. For the number of the filters and for the nodes in the fully connected layers, they also searched through a chosen number of combinations.

3. They finally picked the structure described in section 3.1.4 according to the optimized hyperparameters.

The pseudocode in Algorithm 1 describes the optimization process for DeepSpCas9, and hence for CRISPRon.

Algorithm 1 Algorithm for Optimizing Hyperparameters for DeepSpCas9 and CRISPRon

Input: A dataset D, A number $M = 3$, The grid searching algorithm A
Output: A optimized convolutional neural network architecture S^*
Set $S^* = Null$
Generate variables V_1, V_2, V_3
for $K_1 \in [2, 4]$,$K_2 \in [5, 6]$,$K_3 \in [7, 9]$ **do**
    for $F_1 \in [20, 100, step10]$,$F_2 \in [50, 70, step10]$,$F_3 \in [20, 100, step10]$ **do**
        for $FC_1 \in [80, 120, step20]$,$FC_2 \in [40, 80, step20]$ **do**
            Use A to search the best hyperparameters (K_i^*, F_i^*) where $i = 1, 2, 3$, and FC_1^*, FC_2^*, based on D
            Set $V_i = (K_i^*, F_i^*)$, where $i = 1, 2, 3$
        end for
    end for
end for
for $\forall V_i$ **do**
    Add convolutional type (kernel size K^*, filter number F^*) and pooling layer to the new convolutional block.
end for
Add the convolutional block to architecture S^*
Add a fully connect layer with FC_1^* neurons to architecture S^*.
Add a fully connect layer with FC_2^* neurons to architecture S^*.
Return S^*

The following key points can be helpful in understanding the algorithm.

- Kim et al. fixed the $M = 3$ as shown in the inputs of Algorithm 1 which means that they fixed the **number of convolutional type (The different types of parallel convolutional filters with different kernel size, and each of them can have many filters)** as 3

- The grid searching algorithm, A, is used for searching the best hyperparameters (the kernel size and the number of filters) for each convolutional type of the only

convolutional block (each convolutional block has one convolutional layer consisting of multiple convolutional types and corresponding pooling layers) they used, and the hyperparameters FC_1, FC_2 (neurons) for the two fully connected layers after the convolutional block.

- The red highlighted parts are used to optimize the structure of the fully connected part of the model.

Although this search procedure allows Kim et al. to explore the hyperparameter space of the model structure to some extent, there are several limitations that constrained the model structure to be optimized.

1. They only used one convolutional block and may miss features that can be extracted by a deeper convolutional layer.

2. They did not search for the number of convolutional blocks in the first layer. More convolutional filters with different kernel sizes may lead to a higher expressiveness of the model, and hence may improve the model performance if the training data are sufficient.

3. They searched the filter kernel size in a very small range, where the kernel size ranges from small to medium, and did not search for a kernel with a larger size, which can capture the global features from the longer sequence segment.

4. The total search space is limited, and the random search algorithm they used for hyperparameter searching is not good enough to handle an extensive search space.

In conclusion, Kim et al. have made some explorations of the structure of DeepSpCas9, but these explorations are limited. Xi et al. used more high-quality data to train their CRISPRon model and added ΔG_B, an important feature, to the hidden layer of the model. However, they took the structure directly from DeepSpCas9 without any further optimization. The structure of the model is

critical to its performance. Therefore, further optimization of the structure of the CRISPRon model is necessary.

4.2 The Adaptive CRISPRon Algorithm

As mentioned in the previous section, we need to further optimize the CRISPRon model architecture. Suppose there are L layers and each layer have N possibilities of neurons. Although the search space complexity is $O(N^L)$, for a fully connected neural network model with a small fixed number of hidden layers and range of possibilities of neurons, it is possible to search the hyperparameters of the number of neurons for each layer, as shown in the red highlights of Algorithm 1. However, when the number of neural network layers increases, the search space will expand rapidly. To make matters worse, for a convolutional neural network with multiple convolutional types per convolutional block, the number of convolutional types used in each convolutional layer T, the size of convolutional kernels for each convolutional type K, and the number of filters for each convolutional kernel F should be taken into account. Suppose there are C convolutional layers, and L fully connected layers, the space complexity will be $O((TKF)^C N^L)$. There are many possibilities of T, K, and F, so it is very difficult to search for the hyperparameters to optimize the overall structure.

In order to make it possible to optimize the overall structure of the model to some extent, my algorithm will be based on an important assumption.

Assumption 1 *The optimal hyperparameters of each layer of convolutional neural network have Markov property to a certain extent. Their values depend heavily on the output of the previous layer.*

The definition for Markov property is:

For any positive integer n and possible states $i_0, i_1, \ldots, i_n$ of the random variables,

$$P\left(X_n = i_n \mid X_{n-1} = i_{n-1}\right) = P\left(X_n = i_n \mid X_0 = i_0, X_1 = i_1, \ldots, X_{n-1} = i_{n-1}\right) \tag{4.1}$$

In other words, the assumption means that, for convolutional blocks, the optimized value of the hyperparameters of each convolutional block only depends on the features extracted from the formerly convolutional block (or input layer if it is the first hidden layer). For example, the best hyperparameters (the kernel size, number of convolutional type, and the number of filters) of the first convolution layer only depend on the input sequences, and the best hyperparameters of the second convolutional layer only depend on the output of the first pooling layer, and so on. This assumption allows me to optimize the structure of the model layer by layer, and thus greatly reduces the search space. The space complexity will be $O(TKFC + NL)$ now.

Moreover, for sgRNA prediction, the input sequence length is limited. Therefore, the maximum number of convolutional types used in each convolutional layer T and the size of convolutional kernels for each convolutional type K will both be limited. The maximum number of convolutional blocks C will be limited as well, since the convolutional layer and pooling layer in the next convolutional block will reduce the dimensions of the output from the last block, and this process is recursively layer by layer, from maximum dimension K to dimension 1.

The input dimension is only 30nts for CRISPRon, which means the maximum number of T and K are 30. The number of convolutional blocks C will be less than 5 depends on the kernel size of the filters, since the average pooling layer in CRISPRon will reduce the dimensions of the output from the last block at least by half. My algorithm will focus on optimizing the structures of the convolutional blocks, and hence the structure of the fully connected layers are kept the same as in CRISPRon without further optimizing.

To optimize the convolutional blocks in CRISPRon, I propose an algorithm, the Adaptive CRISPRon Algorithm, to optimize the structure of the convolutional blocks in CRISPRon as follows:

1. Keep the structure of the fully connected layers of CRIPSRon, the convolutional blocks will be added before these layers

2. Set $n = 1$, where n is the n^{th} block currently under optimization and prepared to be added to the model.

3. Generate M variables, where M is the max number of parallel 1D convolution filter types with different kernel sizes at the first block. It can be only chosen from 1 to 30 (since there are only 30nts for the input sequence for CRISPRon)

4. For each of the variables, set a max kernel size. It will be 30 for the first convolutional block, and for n^{th} convolutional block other than 1^{st}, it depends on the output of $(n-1)^{th}$ the block.

5. Use an optimizing algorithm (I used the Tree-structured Parzen Estimator algorithm) to find the optimized structural hyperparameters on a dataset D. For each variable at the n^{th} block, the kernel size search range is from 0 (0 means to ignore this type of filter forever on) to the max kernel size of the variable, and the search range of the number of filters is pre-determined (I searched from 20 to 140 with 20 steps). Input the found hyperparameters to the M variables generated.

6. For each variable, check the optimized value. If it is not equal to 0, then add a type of convolutional filter to the next convolutional layer according to the variables (the optimized structural hyperparameters). If it is 0, then stop adding any convolutional layer or pooling layer for this type to any of the convolutional blocks, forever on.

7. Add the corresponding pooling layer for each of the convolutional type added.

8. Reset the variables, and also the number M. M is the number of parallel 1D convolution filter types at the newly added convolutional layer.

9. Reset $n = n + 1$, and the max kernel size to the length of the output of the newly added pooling layer.

10. Run steps 4 - 9 recurrently until a new convolutional block cannot be added (all parallel types abandoned because of step 6, or the outputs of the previous block for all types are 1).

The pseudocode of the algorithm is presented in Algorithm 2. Notice that The blue shadowed places are different from Algorithm 1.

Algorithm 2 Adaptive CRISPRon Algorithm

Input: A dataset D, A number M where $M \in [1, 30]$, A fully connected neural network structure S, A search range R, the Tree-structured Parzen Estimator algorithm A, the dimension of dataset L
Output: A optimized convolutional neural network architecture S^*
Set $n = 1$
Set $S^* = S$
while $M \neq 0$ **do**
    Generate M variables $V_1^n \dots V_m^n$, and M cooresponding variables $V_1^{n,max} \dots V_m^{n,max}$
    Initialize $\forall O_1^n \dots O_m^n = Null$
    If $n = 1$**:** Set all of $V_1^{n,max} \dots V_m^{n,max}$ to L. **Else:** Set based on $O_1^{n-1} \dots O_m^{n-1}$
    for $K_i \in [0, V_i^{n,max}]$where $i = 1 \dots m$ **do**
        for $F_i \in R$ where $i = 1 \dots m$ **do**
            Use A search best (K_i^*, F_i^*) based on D and S^*
            Set $V_i^n = (K_i^*, F_i^*)$
        end for
    end for
    Initialize $Counter = 0$
    for $\forall V_i^n$ **do**
        If $K^* \in V_i^n \neq 0$ **do**:
        Add convolutional type (kernel size K^*, filter number F^*) and pooling layer to n^{th} convolutional block.
        $Counter = Counter + 1$
        Else: Continue
    end for
    Reset $M = Counter$
    Add the n^{th} block, before fully connected structure S, to architecture S^*
    $n = n + 1$
    Set $O_1^{n-1} \dots O_m^{n-1}$ to the output dimensions of the pooling layer for each V_i^n
end while
Return S^*

There are four notable differences with Algorithm 1 worth highlighting:

- I used the TPE optimization algorithm rather than grid search.
- The number of convolutional types of the first convolutional block is no longer fixed as 3. It can be 1 to M (depending on the outcome of the TPE algorithm), where M is the max value that is chosen manually from 1 to 30, which means the TPE algorithm may explore a bigger space.
- Unlike Kim et al. using only one convolutional block, I added a while loop that iteratively for searching each convolutional block with the Markov property assumption that the optimized value of the hyperparameters of each convolutional block only depends on the features extracted from the formerly convolutional block (or input layer if it is the first convolutional block)
- There is a while loop which will terminate when the descendants of all convolutional types of the first convolutional block stop growing deeper.

In general, compared with the model optimization method of Kim et al., Adaptive CRISPRon Algorithm can explore a larger hyperparameter space, which is conducive to finding the global optimal hyperparameter, so as to realize the optimization of model structure.

Chapter 5

Experiments and Results

Contents

5.1 Data Analysis

As mentioned in Chapter 3 subsection 3.2.1, Xi et al. obtained the high-quality gRNA activity data for 10,592 gRNAs through experiments. They found that the frequency of indel (on-target activity) of gRNAs on days 8 and 10 was highly associated. Therefore, they first used the average gRNA activity of days 8 and 10 for the development of two models: pre-CRISPRon-V0 and pre-CRISPRon-v1. The day 8 and day 10 indel frequency were later used to test and compare with different models. The histogram of average indel frequency distribution of these 10,592 gRNAs is shown in figure 5.1.

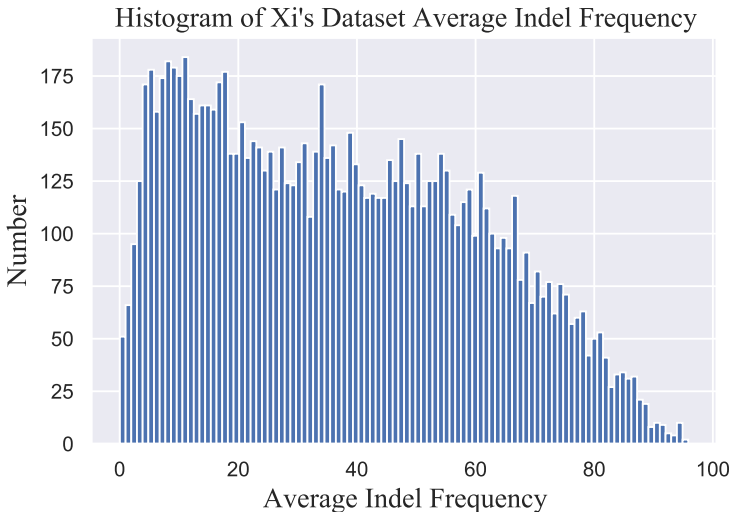

Figure 5.1: The Histogram of Xi's Dataset Average Indel Frequency

It can be seen that the value range of indel frequency is from 0.04 to 95.85, because the result in formula 3.1 is in the form of percentage. In addition, it can be seen that this distribution is not balanced. Therefore, when using them for training, especially when choosing to split the training set and the validation set, it is necessary to ensure that the data distribution is relatively balanced. The data preprocessing methods will be discussed in the next section.

After they trained the above two models, Xi et al. combined their dataset with Kim et al.'s by linear rescaling on the common 30mer sequences that are found in both theirs and Kim et al.'s dataset (in 2019), resulting in a dataset of 23,902 gRNAs. This data is used to train and test CRISPRon-V0 and CRISPRon. It is also used in my experiments to optimize the structure of CRISPRon and to train Adaptive CRISPRon (both version 0 and version 1, see 5.5). The histogram of the average indel frequency distribution of these 23,902 gRNAs is shown in figure 5.2. Again, the distribution is not balanced and needs careful pre-processing. However, the range now runs from -4.41 to 95.10. Scores less than 0 are due to rescaling.

Note that both the Xi et al. data set, and the combined data set, were not published. I obtained these data by contacting Xi et al.

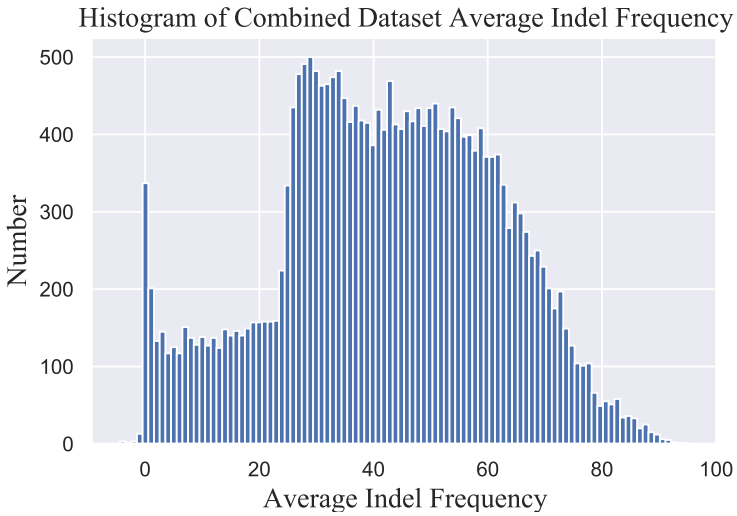

Figure 5.2: The Histogram of the Average Indel Frequency of the Combined Dataset

In addition to the above two data sets, this paper also uses some external data sets. These datasets are only used for model testing and comparison and do not participate in model training. These data also play a similar role in the CRISPRon test by Xi et al. They are from: Doench et al. (in 2014 and 2016) [7, 53], Xu et al. (in 2015) [47], Chari et al. (in 2015) [58], Hart et al. (in 2015) [59], Wang et al. (in 2019) [11], Kim et al. (in 2020)[54]. The data is published and can be downloaded from relevant papers. For convenience, from now on I will use the first author name as the name of the data set.

5.2 Data Pre-processing

5.2.1 ΔG_B Biological Feature Computation

An important feature used in CRISPRon model training is RNA-DNA binding energy ΔG_B. This feature is not provided in the data of Xu et al. and needs to be calculated through Alkan et al.'s CRISPRoff pipeline [1]. The RNA-DNA

[1]I used the CRISPRoff (version 1.1.2) which can be downloaded from `https://rth.dk/resources/crispr/crisproff/download`

binding energy ΔG_B is calculated by equation 5.1.

$$\Delta G_B = \delta_{PAM} (\Delta G_H - \Delta G_U - \Delta G_O) \tag{5.1}$$

Where ΔG_H is the position-weighted binding energy between gRNA and target DNA, ΔG_O is the free energy of the DNA duplex, ΔG_U is the folding energy of gRNA alone and δ_{PAM} is a correcting factor according to the type of PAM sequence. Therefore, given a gRNA sequence, the CRISPRoff pipeline outputs a file with four features: ΔG_H, ΔG_O, ΔG_U, and ΔG_B, in which only ΔG_B is used. The range of ΔG_B of the combined dataset is from -13.11 to 51.20, and the distribution is shown in figure 5.3. The distribution is fairly uniform, with most sgRNA scores clustered between 20 and 30.

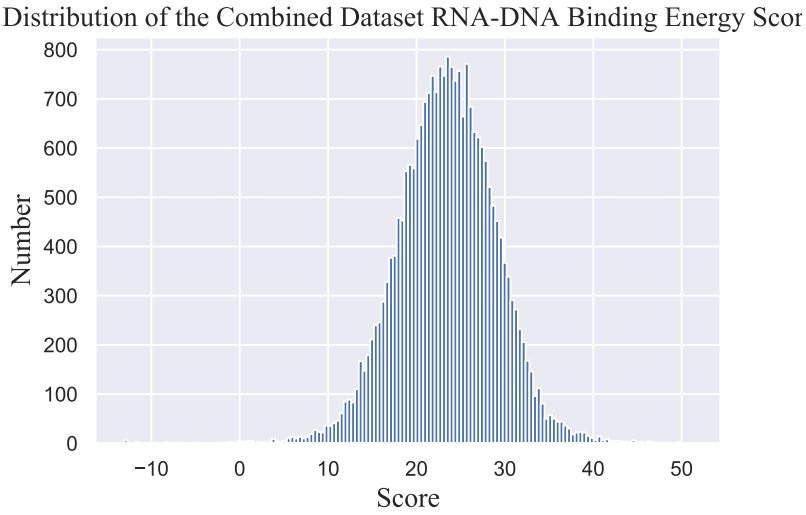

Figure 5.3: Distribution of the Combined Dataset RNA-DNA Binding Energy Score

5.2.2 Training Data Split

In this paper, the training data (Xi generated and combined dataset) are divided into six splits with reference to the classification method mentioned in the paper by Xi et al. before training. The size difference of these six parts of data is within 1 gRNA. The data is separated as follows:

1. Use SciPy's pdist function to calculate the pairwise Hamming distance between all gRNAs based on their 30mer one-hot encoded sequences (gRNA plus context).

2. A list of all gRNAs with a Hamming distance of 8 or less in the one-hot space, which translates to a sequence difference of 4 nt, was considered "similar" to each gRNA x.

3. gRNAs that were similar to at least one other gRNA in the data set were the first to be randomly distributed in the partitions; when a gRNA was assigned to a partition, all gRNAs that were similar to it (and recursively those that were similar to these gRNAs) were also added to the same partition.

4. After exhausting all similar gRNAs, the remaining gRNAs were divided into three subsets based on their efficiency (inefficient: up to the percentile of efficiency 25 (25p), medium efficient: from 25 to 75p, and highly efficient: above 75p), and the gRNAs in these three subsets were distributed to the partitions pseudo-randomly by assigning a balanced amount of inefficient, medium efficient, and highly efficient gRNAs to each partition until the predesigned size was reached.

Note that when training and testing the CRISPRon-V0 and my Adaptive CRISPRon-V0, the data annotated as the test set in Kim et al. (2019) (526 in total) of the combined dataset, was treated as an initial partition before any splitting and any gRNAs similar to the initial group in the combined are assigned to this partition. In other words, the independent test set for CRISPRon-V0 and Adaptive CRISPRon-V0 is the test set of Kim et al. (2019) after train and test set preprocessing (see section 5.2.3).

In addition to the above segmentation methods, this paper also uses sklearn's train-test-split tool to randomly split the training set for training. The performance of the CRISPRon-Extra model trained in this way will be slightly worse than that of CRISPRon, which will be mentioned in section 5.3.

5.2.3 Test Data Pre-processing

The gRNA similar to the training data should be removed from all the test data sets to ensure that they are completely independent. All test sets are preprocessed as follows:

1. Calculate the pairwise hamming distance between the gRNAs in the test and training datasets with the one-hot encoded 20 nt protospacer and cdist function in Scipy.

2. gRNAs that had a sequence difference of less than three nucleotides (nt) and a Hamming distance of less than six in the one-hot space were eliminated.

The preprocessing method of the test set (and the separation method of the training set) is completely consistent with that of Xi et al., in order to ensure that the models in my paper and Xi et al.'s paper are comparable.

In addition, external data sets require some additional preprocessing during selection. All data sets can be divided into two categories: one is used to complete the study of gene loss, including Xu (2015), Hart (2015), and Doench (2014-2016); the other is based on the Indel frequency, including: Kim (2019–2020), Wang (2019), Chari (2015), and Xi (2021). The preprocessing of the data sets involved the elimination of gRNAs that fit one of the following descriptions. Notice that points 2, 4, and 7 are only for the studies of gene loss.

1. Not in hg38[2].

2. GENCODE annotations did not match target gene expression.

3. High inter-experimental variability in efficiency, over the threshold:upper quartile + 1.5 × variance interquartile range.

4. The gene with less than ten gRNA.

[2]hg38 is the ID used for GRCh Build 38 in the context of the UCSC Genome Browser.

5. The corresponding PAM is not 5'-NGG-3'.

6. Expressed from a tRNA system.

7. Targeting the final 10% of a combined gene's identified coding sequences.

Note that for Doench (2014), I only used human cell data. For Doench (2016) the gRNAs marked for low early time point (ETP) are eliminated. For Wang (2019), gRNAs without any context information are also eliminated.

The external datasets now have the following number of sequences after the processing mentioned above: Kim (2020): 8742; Xu(2015): 971; Chari (2015): 1,224; Hart (2015): 4001; Doench (2014): 781; Doench(2016): 2145; Wang (2019): 55,022;

The preprocessing method of external data is also completely consistent with that of Xi et al. The preprocessed external data can be obtained by asking Xi et al.

5.3 Model Reproduction

This paper mainly reproduced the results for four models: pre-CRISPRon-V0 and pre-CRISPRon-V1 with Xi et al.'s dataset, CRISPRon-V0 and CRISPRon with the combined dataset. I also trained a CRISPRon-Extra by randomly splitting the same training set as CRISPRon rather than the carefully split stated in subsection 5.2.2. The other four models' (DeepHF, DeepSpCas9, Azimuth, and DeepSpCas9Variants) results are reproduced by Xi et al. and their codes are directly available in GitHub link in the respective papers.

The codes for CRISPRon[3] cannot be directly used to reproduce the results, since the code is embedded into a software, takes different inputs, and output predictions of a given gRNA instead of Spearman's correlation. Therefore, I modified the input and output interfaces and the functions for read files and data preprocessing. The model structure constructed by Keras (TensorFlow API) is unchanged for all the reproduced models, since they adopted the same structure but different training data. The structure is shown in figure 5.4.

[3] `https://github.com/RTH-tools/crispron`

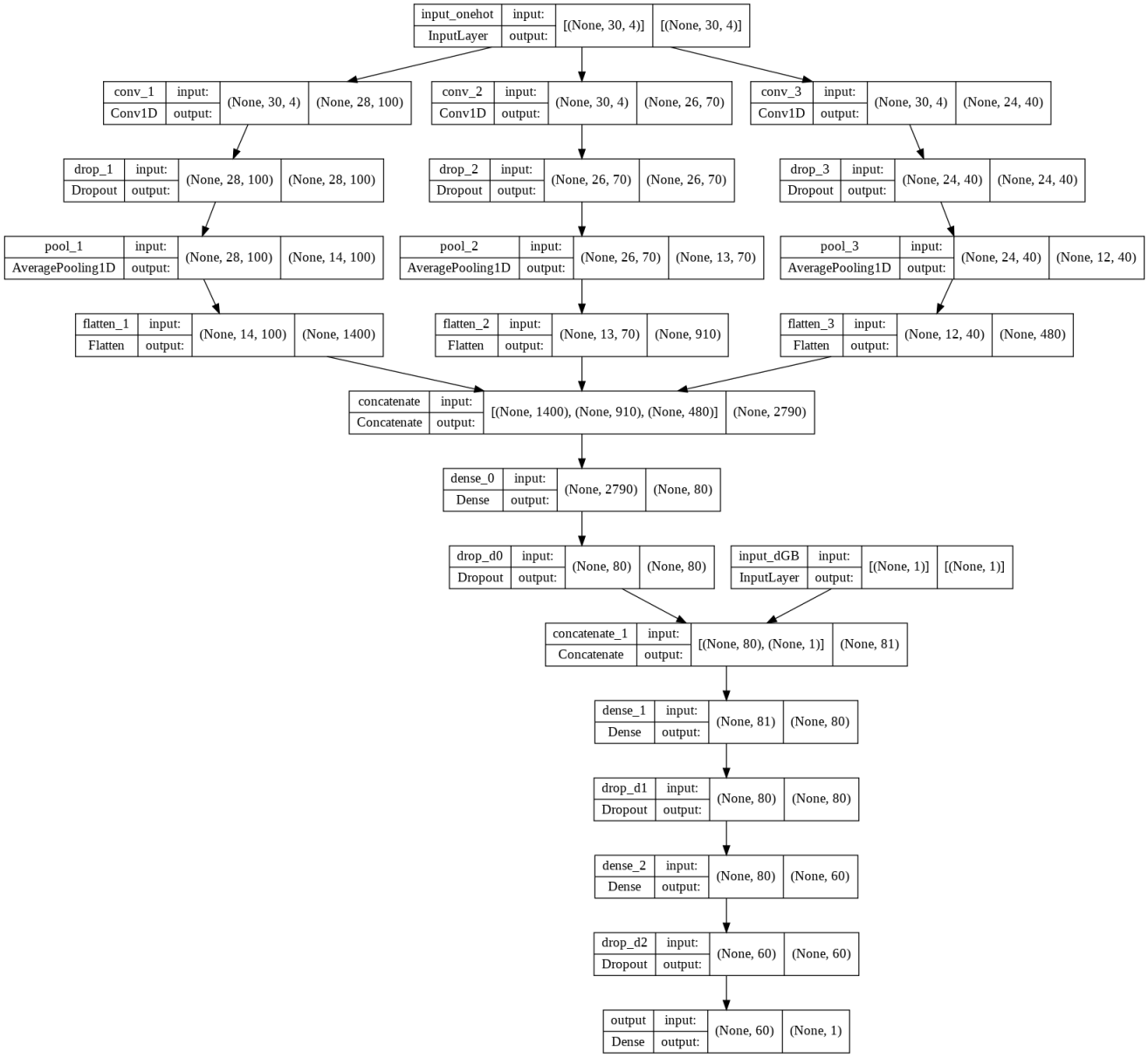

Figure 5.4: The Structure of Reproduced Models

The structure of the model is the same as the structure in figure 3.4. Each input to the model is a one-hot encoded sequence vector with length 30, and 4 dimensions to represent each nucleotide. The input first passes through three parallel filter types with kernel size 3, 5, and 7. Then, it passes through the corresponding dropout layer, pooling layer and flatten layer, and then be concatenated in to one long vector

with length 4190. After the first dense layer (and the corresponding dropout layer), the ΔG_B is then concatenated with the output to pass through two more dense layers and dropout layers to get a final value. This value is then used to calculated Spearman's correlation with the true value if the model is used for prediction, or to compute the MSE and propagated back to update the weight if the model is training.

All reproduced models use the same set of hyperparameters when training, which is consistent with that used by Xi et al. The learning rate of the model is 0.0001, the optimizer is Adam and the batch size is 500. The model also adopts the early stop technique to mitigate the overfitting problem. The training epoch is set as 3000 while the performance on the validation set did not improve for 100 consecutive epochs, the training was terminated, and the MSE best performing model on the validation set was retained. When cross-validating, the training of each fold was done ten times using random seeds, and the best model from the ten was picked.

The first two reproduced models are pre-CRISPRon-V0 and pre-CRISPRon-V1. As mentioned in subsection 5.2.2, the dataset generated by Xi et al. were divided into 6 partitions. pre-CRISPRon-V0 is trained with a 5-fold cross-validation, while the partition 6^{th} is used as an independent internal set to measure performance together with external data sets. After training and measuring the pre-CRISPRon-V0, all six partitions are used to train the pre-CRISPRon-v1 model with six-fold cross-validation, and the model is tested on the external independent test sets. The comparison between the performance of the reproduced version with the original version for these two models is shown in table 5.1.

As can be seen in the table, the results of the repeated model are almost the same as those of the original model, except for slight differences in some data sets, which may be caused by the selection of random seeds. The pre-CRISPR-V1 is generally better than the pre-CRISPR-V0, since more high-quality data are used in training for pre-CRISPR-V1. Note that for the pre-CRISPRon-V1 model, the data from Xi et al. cannot be used for model evaluation because they have already been used for model training.

	Pre-V0 (Ori)	Pre-V0 (Repr)	Pre-V1 (Ori)	Pre-V1 (Repr)
Chari 2015	0.436	0.434	0.442	0.439
Doench 2014	0.608	0.606	0.615	0.615
Doench 2016	0.330	0.327	0.354	0.354
Hart 2015	0.468	0.467	0.469	0.467
Kim 2019	0.732	0.732	0.737	0.734
Kim 2020	0.346	0.345	0.346	0.346
Wang 2019	0.722	0.721	0.727	0.727
Xu 2015	0.564	0.563	0.567	0.565
Xi 2021 day10	0.825	0.825	NA	NA
Xi 2021 day8	0.800	0.797	NA	NA

Table 5.1: The Spearman's Correlation of Reproduced and Original Version of pre-CRISPRon-V0 and pre-CRISPRon-V1.

The next two models to be reproduced are CRISPR-V0 and CRISPRon. The combined data set of Xi (2021) and Kim (2019) was divided into 6 partitions. CRISPRon-V0 is trained with a five-fold cross-validation, while the 6^{th} partition is used as an internal independent set for measuring the performance together with the external datasets. After CRISPRon-V0 is trained and measured, the six partitions are used to train the CRISPRon model with 6-fold cross-validation, and the model is tested on the external independent test sets. The comparison between the performance of the reproduced version with the original version for these two models is shown in table 5.2

Similarly, the replicated model and the original model produced almost identical results, except for a few minor differences. Compared to the two pre-CRISPRon models, CRISPRon and CRISPRon-V0 were tested on the average Indel frequency of DAY8 and DAY10 of the combined data set. The CRISPRon results are far better than the performance of other existing models on the same dataset (the results are reported in Supplementary Table 2 in their paper [12]), which is the main contribution of Xi et al. Note that for CRISPRon the Kim et al. (2019) and Xi et al. (2021) datasets have been used for training and cannot be used to measure model performance.

	V0 (Ori)	V0 (Repr)	CRISPRon(Ori)	CRISPRon (Repr)
Chari 2015	0.464	0.464	0.464	0.464
Doench 2014	0.676	0.676	0.681	0.680
Doench 2016	0.439	0.436	0.448	0.448
Hart 2015	0.500	0.498	0.499	0.498
Kim 2019	0.805	0.805	NA	NA
Kim 2020	0.451	0.451	0.462	0.461
Wang 2019	0.682	0.681	0.683	0.683
Xu 2015	0.583	0.581	0.577	0.575
Xi 2021 day10	0.810	0.810	NA	NA
Xi 2021 day8	0.779	0.779	NA	NA
Combined Avg	0.804	0.804	NA	NA

Table 5.2: The Spearman's Correlation of Reproduced and Original Version of CRISPRon-V0 and CRISPRon.

In addition to these four models, I trained a CRISPR-Extra model. This model is only used to compare the impact of different data segmentation on the model. I get CRISPRon-Extra by randomly splitting the combined data into training and validation sets using the train-test split function in Sklearn and training the model with a five-fold cross-validation. Its performance is worse than that of CRISPRon-V0, by the least 2 percents on all datasets, which represents the importance of data pre-processing.

5.4 Model Structure Optimizing

5.4.1 Hyperparameter Search Tool

In order to be able to optimize the model structure using Algorithm 2, the search tool should be selected to find the best hyperparameters (line 11 of Algorithm 2). I chose to use the tool Hyperopt. Hyperopt is a distributed asynchronous hyperparameter optimization tool and one of the most commonly used Bayesian optimizers. Multiple optimization algorithms including random search, simulated annealing and Tree-structured Parzen Estimator (TPE) algorithm are embedded in Hyperopt. Hyperopt is a more advanced, modern and better maintained optimizer than other tools such

as Bayesopt, and is one of the most commonly used optimizers to implement the TPE algorithm, which is used by Algorithm 2 to search for the structure of the model.

The process of using the Hyperopt tool consists of five steps:

1. Define the objective function. When performing this step, note that the input to the objective function must be a Hyperopt-compliant dictionary.

2. Defining the hyperparameter space. Hyperparameters combination can be input into the defined objective function in step 1.

3. Define the specific process of optimizing the objective function. In Hyperopt, the basic feature that can be used for optimization is called fmin. In fmin, one can select the algorithm used for the search. There are two options: TPE.suggest and Rant.suggest. The former refers to the TPE method, and the latter refers to the random grid search method. One also need to set the number of evaluation the tool will search, and then stop.

4. Define validation functions (not always necessary).

5. Perform the optimization process.

Note that Hyperopt only supports finding the minimum value of the objective function, but does not support finding the maximum value. Therefore, when the defined objective function is some positive evaluation metric (such as Spearman's correlation), the value of the metric needs to be negated. If the objective function is defined as a negative loss, one needs to take the absolute value of the negative loss.

Hyperopt has an early stop function $earlystopfn()$. When the loss does not fall many times in a row, it can cause the algorithm to stop early. Due to the high randomness of the Bayesian method, it may take a lot of iterations to find the optimal solution when the sample size is insufficient. Therefore, in general, the value of the parameters in $earlystopfn()$ cannot be set too low.

5.4.2 Implementation of the Adaptive CRISPRon Algorithm

The following input is required to execute Algorithm 2: A dataset D, A number $M \in [1, 30]$, A fully connected neural network structure S, A search range R, and the dimension of the dataset L.

In order to compare with the CRISPRon model, I selected the combined dataset of Xu et al. and Kim et al. as input data D, where the data annotated as the test set in Kim et al. (2019) of the combined dataset, was not participated in the structure optimizing and treated as an independent test set as mentioned in subsection 5.2.2. A number M represents the number of convolutional type is set as three, which is the same as in Algorithm 1, the algorithm Kim et al. used for optimizing DeepSpCas9 (see Chapter 4 for details). This is to ensure that the optimized structure is not too different from the CRISPRon structure for comparison. At the same time, if M is too high, the search space may be large and the best hyperparameter searching will be time-consuming. (However, changes in the value of M may further improve the model's performance, see section 6.1). S is kept the same as the fully connected neural network structure in CRISPRon and the search range R of number of filters for each convolutional type is from 20 to 140 with 10 steps. The dimension of dataset L is fixed as 30.

The algorithm is executed using the Hyperopt tool mentioned in subsection 5.4.1. According to Algorithm 2, the Hyperopt tool needs to be run several times until the new convolutional layer can no longer be added. Also, for each run, the Hyperopt objective function and search space need to be redefined based on the previous results. The objective function of Hyperopt at each time is the current model being optimized (the model is optimized layer by layer, from the first convolutional layer), and the target search space, as shown in Algorithm 2, shrink for each run (for the first run, the search space is the kernel size, from 1 to 30, and the number of filters, from 20 to 140 with 10 steps, at the first convolutional layer). The algorithm used in searching is TPE and the number of evaluation for each run is 200, with the early

stopping number 50. The hold-out validation method, rather than five-fold cross validation, is used during structure optimizing for lowing the searching time cost.

5.4.3 The Adaptive CRISPRon Structure

For a given input in subsection 5.4.2, Hyperopt ran a total of three times until termination. The optimal three kernel sizes searched in the first convolutional layer are 5, 6 and 24, and the number of filters are 140, 60 and 20.

For the second run, the best kernel size of the second (kernel size 6 at the first layer) and third (kernel size 24) convolutional types are zero, and hence the descendants of these two convolutional types stop growing. For the first convolutional types, the optimal kernel size of the second convolutional layer is 3 and number of filters is 60.

For the third run, the optimal kernel size of the first convolutional type in the third convolutional layer is 5 and the number of filters is 60.

After the third run, the output dimension of the pooling layer of the first convolutional type is 1, and hence the descendants of these all convolutional types stop growing.

The optimized structure is shown in figure 5.5. The fully connected layer of the new model structure is consistent with the structure of CRISPRon, but there are two obvious differences in the structure of the convolutional layers.

- Compared with CRISPRon, the convolutional layer structure of the new model is not symmetric. The first convolution type has two descendants, and the second and third convolution types don't have any descendants.

- The convolution kernel size of one of the convolution types in the first layer of the new model is 24, which is very large and almost covers the entire gRNA sequence length (30nt).

These differences indicate that for some extracted features, it is necessary to carry out further feature extraction (adding descendants for corresponding convolution

types), and it is also necessary to carry out global feature extraction for gRNA sequences (high kernel size).

In addition, it is worth noting that a convolution type has the same kernel size as CRISPRon. This may indicate that for gRNA, five consecutive nucleotides contain some special features, so that the convolution type with kernel size of 5 can extract the key features.

5.5 Adaptive CRISPRon Training and Testing

According to the Adaptive CRISPRon structure obtained in section 5.4, I trained two models, Adaptive CRISPRon-v0 and Adaptive CRISPRon. Similarly to the way Xi et al. trained CRISPRon-V0 and CRISPRon, my two models share the same structure while using different splits of data sets for training. Adaptive CRISPRon-V0 is trained with a five-fold cross-validation on the partitioned data 1^{th} to 5^{th} of the combined data set of Xi (2021) and Kim (2019), while the partition 6^{th} (the test set indicated by Kim et al.) is used as an independent internal set to measure performance together with the external datasets. After Adaptive CRISPRon-V0 is trained and measured, the six partitions are used to train the CRISPRon model with 6-fold cross-validation, and the model is tested on the external independent test sets. All the hyperparameters used in training are completely consistent with CRISPRon and CRISPRon-V0. The learning rate of the model is 0.0001, the optimizer is Adam, the batch size is 500, the training epoch is 3000 and the early stopping number is 100. When cross-validating, the training of each fold was done ten times using random seeds, and the best model from the ten was pick.

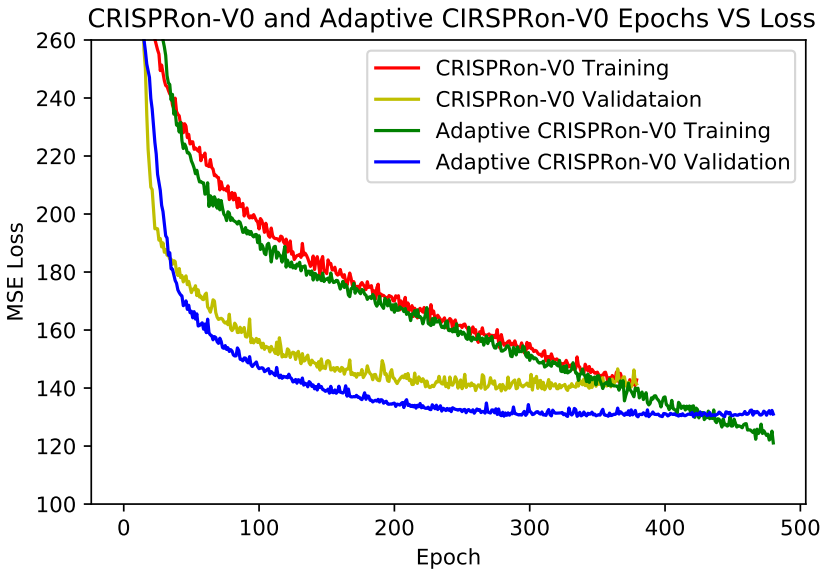

Figure 5.6: CRISPRon-V0 and Adaptive CIRSPRon-V0 Epochs VS Loss

The epochs versus loss graph of both validation set and training set for Adaptive CRISPRon-V0 is shown in figure 5.6, drawn by Matplotlib from the log exported by tensor board. For comparison, the figure shows that for CRISPRon-V0 as well. As can be seen from the figure, the loss of Adaptive CRISPRon-V0 on the validation set starts to be lower than that of CRISPRon-V0 on the validation set at the 35^{th} epochs, and the loss gap continues to expand after that. CRISPRon-V0 has a minimum loss of 140.5 on the validation set (140.7 as reported by Xi et al.), while Adaptive Crispron-V0 has a minimum loss of only 130.9 on the validation set. Although there is no significant difference between the two models in terms of the loss on the training set before Crispron-V0 stops early, it can still be seen that Adaptive CRISPRon-V0 has a lower loss. Another point worth noting is that the early stop time of Adaptive Crispron-V0 is 102 epochs later than CRISPRon-V0. This is as expected, because the model structure of Adaptive CRISPRon-V0 is more complex and requires more time to train.

The performance of Adaptive CRISPRon-V0 is tested on internal independent dataset and other external datasets, and the results, with comparison with CRISPRon-V0, in shown in table 5.3. As can be seen, Adaptive CRISPRon-V0 outperforms CRISPRon-V0 on 9 of the 11 datasets tested, with only Hart (2015) and Chari (2015) slightly lower than CRISPRon, where the difference is within 0.001. The difference between the two models in the prediction of Indel frequency on the eighth and tenth day on Xi (2021) dataset is the largest. Adaptive CRISPRon-V0 has reached the lead of 21 percentage and 14 percentage respectively. The differences in performance on the Chari (2015) and Doench (2014, 2016) datasets are relatively small, with 3 percentage lags, 4 percentage leads and 4 percentage leads, respectively.

	AdaptiveCRISPRonV0	CRISPRonV0
Chari 2015	0.461	**0.464**
Doench 2014	**0.685**	0.681
Doench 2016	**0.452**	0.448
Hart 2015	0.490	**0.499**
Kim 2019	**0.820**	0.805
Kim 2020	**0.471**	0.462
Xi 2021 day10	**0.832**	0.810
Xi 2021 day8	**0.793**	0.779
Wang 2019	**0.690**	0.683
Xu 2015	**0.581**	0.577
Combined Avg	**0.815**	0.804

Table 5.3: The Spearman's Correlation of Adaptive CRISPRon-V0 Compared with CRISPRon-V0

Figure 5.6 and table 5.3 indicates that adopting the Adaptive CRISPRon structure have huge potential to obtain a model has great performance, and surpass that of CRISPRon.

After that, I trained Adaptive CRISPRon using 6-fold cross-validation on the combined dataset.

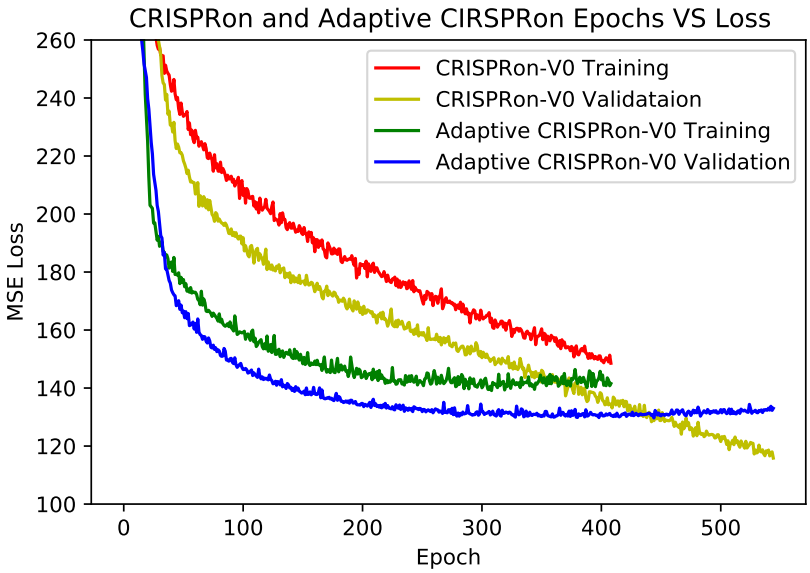

Figure 5.7: CRISPRon and Adaptive CRISPRon Epochs VS Loss

Figure 5.7 shows the epochs versus loss graph of both validation set and training set for Adaptive CRISPRon and CRISPRon. It can be seen that more data leads to more training epochs. CRISPRon stops at about 405 Epochs, while Adaptive CRISPRon requires about 675 Epochs.

In addition, it can be noticed that the loss does not decrease significantly when CRISPRon (also Adaptive CRISPRon) is compared with CRISPRon-V0 (Adaptive CRISPRon-V0), as shown in table 5.2. The MSE for CRISPRon is 139.1 (not reported by Xi et al.) and for Adaptive CRISPRon is 130.1. This is normal, as mentioned earlier, the extra data CRISPRon used for training is only the Kim et al. annotated as the test set (and the gRNAs in combined set that are similar to them), which is the sixth partition used for testing in training CRISPRon-V0 (see section 5.2.2).

Figure 5.7 indicates that Adaptive CRISPRon has better performance than CRISPRon on validation set. For testing the performance and robustness of Adaptive CRISPRon, I run Adaptive CRISPRon on the external datasets and

the results, compared to the models with advanced performance, are shown in table 5.4. As you can see from the table, Adaptive CRISPRon performs extremely

	AdaptiveCRISPRon	CRISPRon	DeepSpCas9	Azimuth	DeepSpCas9variants	DeepHF
Chari 2015	0.461	**0.464**	0.439	0.356	0.252	0.416
Doench 2014	**0.688**	0.681	0.591	NA	0.309	0.653
Doench 2016	**0.449**	0.448	0.418	NA	0.330	0.370
Hart 2015	0.497	**0.499**	0.470	0.381	0.307	0.455
Kim 2019	*cannotbetested*	*cannotbetested*	0.771	0.574	0.389	0.716
Kim 2020	0.476	0.462	0.483	0.344	**0.697**	0.372
Xi 2021 day10	*cannotbetested*	*cannotbetested*	0.717	0.560	0.259	0.755
Xi 2021 day8	*cannotbetested*	*cannotbetested*	0.693	0.547	0.248	0.725
Wang 2019	**0.695**	0.683	0.525	0.454	0.248	NA
Xu 2015	0.581	0.577	0.527	0.429	0.212	**0.613**
Combined Avg	*cannotbetested*	*cannotbetested*	0.727	0.563	0.295	0.738

Table 5.4: The Spearman's Correlation of Adaptive CRISPRon Compared with Advanced Models

well. It leads CRISPRon across almost all data sets. Except for Chari (2015) and Hart (2015), they performed slightly worse, but the difference was within 0.002. What's more exciting is that Adaptive CRISPRon achieves state-of-the-art results on Donech (2014, 2016) and Wang (2019).

Note that Kim 2019, Xi 2021 and the combined dataset of these two, cannot be used to test CRISPRon since they have participated in the training process of the model. However, Adaptive Crispron-V0 also achieves state-of-art performance on these datasets. This revealed that the optimized structure produced better performance than the CRISPRon structure.

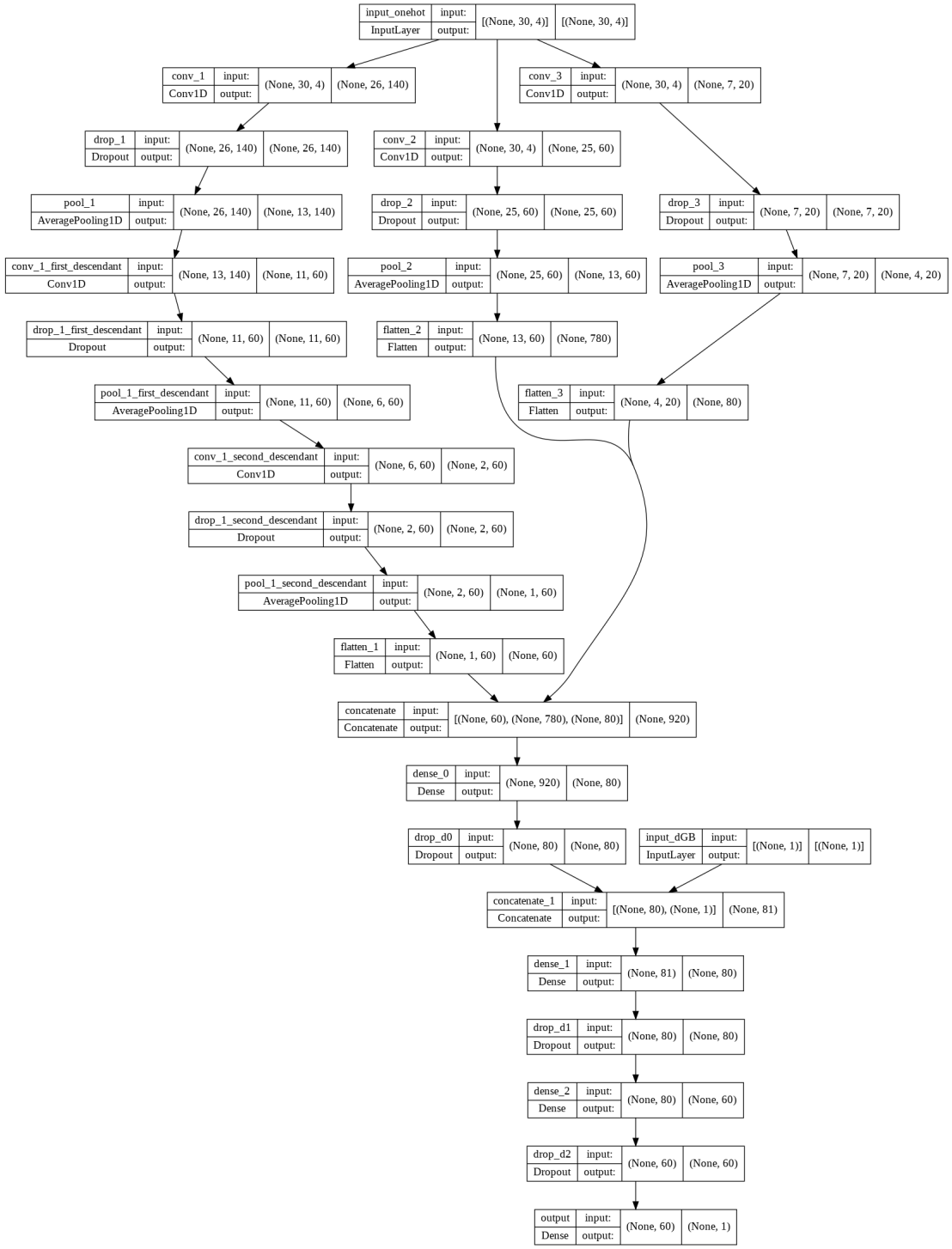

Figure 5.5: The Structure of Adaptive CRISPRon

Chapter 6

Discussion and Conclusion

Contents

6.1 Discussion and Future Work Suggestion

The excellent performance of Adaptive CRISPRon can be used to develop a new tool that can be used to accurately predict the on-target activity of gRNAs, helping biologists design more efficient gRNAs. In addition, the Adaptive CRISPRon Algorithm makes it possible to optimize the model based on the CNN structure. Many models based on CNN structures, especially those used in the field of biological sequence analysis, suffer from trade-off problems of expressiveness and robustness. On the one hand, an overly complex structure may lead to overfitting of the trained model due to insufficient data. On the other hand, an overly simple model structure may fail to capture important patterns from sufficient data due to lack of flexibility, resulting in underfitting. This problem can be alleviated by optimizing the model structure. The adaptive CRISPRon Algorithm that I proposed can provide a certain degree of regularization effect. In the optimization process of the model structure, the convolution kernel size of each convolution type in each layer can be equal to zero. This reduces the complexity of the model to some extent, similar to the

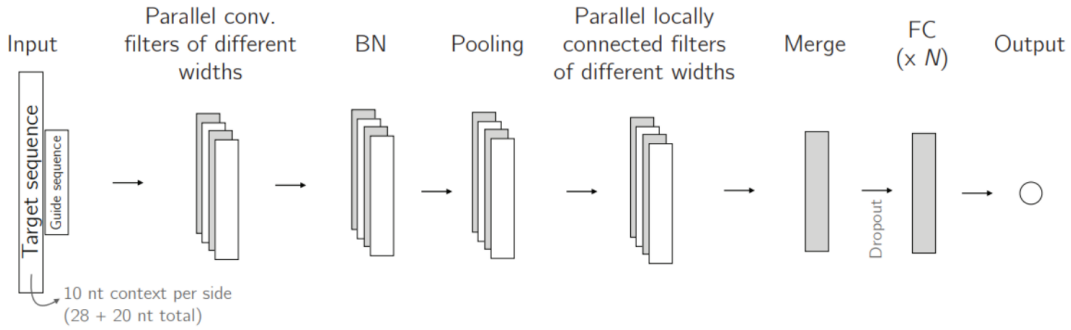

Figure 6.1: Architecture of Metsky et al.'s Model [60]

effect of the dropout technique on the full link layer. Meanwhile, the structure is searched by the optimization algorithm, which can find the most suitable model structure to capture the current data pattern. Therefore, the performance of the structure optimized by the algorithm is generally better than that of the unoptimized structure. For example, Metsky et al. proposed an CNN-base model, although it was used on design of nucleic acid-based viral diagnostics rather than gRNAs. It used parallel filters of different widths (which is the convolutional type), as shown in figure 6.1. They did not use any hyperparameter optimizing algorithm to search the architecture (It may be good if they have enough data, but the high model complexity may lead to overfitting for a small dataset). Using my algorithm can further improve their structure. To be specific, the structure of their model could be one possible outcome case of the Adaptive CRISPRon Algorithm with M input of 30 (48 in their case, since their input dimension is 48nt).

Except the advantages mentioned above, there are still some limitations and some work that can be further engaged. To improve on the current work, there are some suggestions that future researchers can take as references:

1. **Further optimization of CRISPRon via Adaptive CRISPRon Algorithm.** In order to better compare with CRISPRon, the input M of Adaptive CRISPRon is set to 3 in this paper (the value can be from 1 to

30 for CRISPRon). This means that the structure of the search model is limited to a small scope. Future research could try different possibilities for entering the numerical value M, which would most likely further improve CRISPRon performance. However, please note that when the value of M is very small or very large, the search space of the algorithm is small because of the less number of combinations of different convolutional types, so the running time cost is relatively low. However, when M takes an intermediate value, the search space of the algorithm will be large, and the running time cost is relatively high.

2. **Optimize other CNN-based model that is used for gRNA on-target activity prediction.** The Adaptive CRISPRon algorithm has demonstrated its potential on the CRISPRon model. Other CNN-based models, such as DeepCRISPR and DeepSpCas9Variant mentioned in Chapter 3, or some future models can use this algorithm to optimize performance.

3. **Optimize CNN-based models used for other sequence analysis tasks.**The Adaptive CRISPRon Algorithm is not only limited to the task of on-target gRNA activity prediction. It can also be used to optimize the model used for gRNA off-target possibility prediction, or even for other biological sequence analysis tasks such as aforementioned the task of design of nucleic acid-based viral diagnostics, as long as the model is based on CNN.

4. **To develop a better algorithm that can optimize models based on other deep learning structures.** The Adaptive CRISPRon Algorithm is limited to optimizing CNN-based model for sequence analysis. However, the assumption and the idea the algorithm used can be transplanted or help to develop other structure searching algorithm, which can be used on models based on other structures such as RNN or LSTM.

6.2 Conclusion

How to predict the on-target activity of gRNA is a key issue in designing gRNA to improve the efficiency of CRISPR-Cas9 technology. Previous models based on CNN structure have good results for this task, but there is still room for improvement. This paper proposes an algorithm to optimize the model based on CNN structure, which can further improve the performance of the model. The trained model outperforms the original model on several datasets and achieves a state-of-art effect. The algorithm proved the importance of structure of a deep learning model and provides possibility to improve the performance of future models for other biological sequence analyzing tasks, such as off-target gRNA prediction and design of nucleic acid-based viral diagnostics.

References

[1] Barry L Stoddard. "Homing endonuclease structure and function". In: *Quarterly reviews of biophysics* 38.1 (2005), pp. 49–95.

[2] Yang-Gyun Kim, Jooyeun Cha, and Srinivasan Chandrasegaran. "Hybrid restriction enzymes: zinc finger fusions to Fok I cleavage domain." In: *Proceedings of the National Academy of Sciences* 93.3 (1996), pp. 1156–1160.

[3] Yong Zhang et al. "Transcription activator-like effector nucleases enable efficient plant genome engineering". In: *Plant physiology* 161.1 (2013), pp. 20–27.

[4] Yuanwu Ma, Lianfeng Zhang, and Xingxu Huang. "Genome modification by CRISPR/Cas9". In: *The FEBS journal* 281.23 (2014), pp. 5186–5193.

[5] Raj Chari et al. "sgRNA Scorer 2.0: a species-independent model to predict CRISPR/Cas9 activity". In: *ACS synthetic biology* 6.5 (2017), pp. 902–904.

[6] Tessa G Montague et al. "CHOPCHOP: a CRISPR/Cas9 and TALEN web tool for genome editing". In: *Nucleic acids research* 42.W1 (2014), W401–W407.

[7] John G Doench et al. "Rational design of highly active sgRNAs for CRISPR-Cas9–mediated gene inactivation". In: *Nature biotechnology* 32.12 (2014), pp. 1262–1267.

[8] Miguel A Moreno-Mateos et al. "CRISPRscan: designing highly efficient sgRNAs for CRISPR-Cas9 targeting in vivo". In: *Nature methods* 12.10 (2015), pp. 982–988.

[9] Guohui Chuai et al. "DeepCRISPR: optimized CRISPR guide RNA design by deep learning". In: *Genome biology* 19.1 (2018), pp. 1–18.

[10] Hui Kwon Kim et al. "SpCas9 activity prediction by DeepSpCas9, a deep learning–based model with high generalization performance". In: *Science advances* 5.11 (2019), eaax9249.

[11] Daqi Wang et al. "Optimized CRISPR guide RNA design for two high-fidelity Cas9 variants by deep learning". In: *Nature communications* 10.1 (2019), pp. 1–14.

[12] Xi Xiang et al. "Enhancing CRISPR-Cas9 gRNA efficiency prediction by data integration and deep learning". In: *Nature communications* 12.1 (2021), pp. 1–9.

[13] Xenopus Xenbase. 2016. URL: `https://www.xenbase.org/entry/static-xenbase/CRISPR.jsp`.

[14] Thomas Gaj, Charles A Gersbach, and Carlos F Barbas III. "ZFN, TALEN, and CRISPR/Cas-based methods for genome engineering". In: *Trends in biotechnology* 31.7 (2013), pp. 397–405.

[15] Hélène Deveau, Josiane E Garneau, and Sylvain Moineau. "CRISPR/Cas system and its role in phage-bacteria interactions". In: *Annual review of microbiology* 64 (2010), pp. 475–493.

[16] Magdalena Hryhorowicz et al. "CRISPR/Cas9 immune system as a tool for genome engineering". In: *Archivum immunologiae et therapiae experimentalis* 65.3 (2017), pp. 233–240.

[17] Kira S Makarova et al. "An updated evolutionary classification of CRISPR–Cas systems". In: *Nature Reviews Microbiology* 13.11 (2015), pp. 722–736.

[18] Fuguo Jiang, Jennifer A Doudna, et al. "CRISPR-Cas9 structures and mechanisms". In: *Annu Rev Biophys* 46.1 (2017), pp. 505–529.

[19] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[20] Donald E Hilt and Donald W Seegrist. *Ridge, a computer program for calculating ridge regression estimates*. Department of Agriculture, Forest Service, Northeastern Forest Experiment ..., 1977.

[21] RJ Tibshirani. "Regression shrinkage and selection via the lasso". In: (2011).

[22] Abhishek Sharma. "Decision Tree vs. Random Forest – Which Algorithm Should you Use?" In: *Analytics Vidhya* (2020).

[23] Thais Mayumi Oshiro, Pedro Santoro PerezJosé, and Augusto Baranauskas. "How Many Trees in a Random Forest?" In: *International Workshop on Machine Learning and Data Mining in Pattern Recognition* (2012).

[24] Guolin Ke et al. "LightGBM: A Highly Efficient Gradient BoostingDecision Tree". In: *Microsoft Research* (2017).

[25] Wiki. *Support-Vector Machine*. Aug. 2022. URL: `https://en.wikipedia.org/wiki/Support-vector_machine`.

[26] Mariette Awad and Rahul Khanna. "Support Vector Regression". In: *Efficient Learning Machines* (2015).

[27] Sunitha Basodi et al. "Gradient amplification: An efficient way to train deep neural networks". In: *Big Data Mining and Analytics* 3.3 (2020), pp. 196–207.

[28] Maurice Henry Buettgenbach. *Explain like I'm Five: Artificial neurons*. `https://towardsdatascience.com/explain-like-im-five-artificial-neurons-b7c475b56189`. Nov. 2021.

[29] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), pp. 533–536.

[30] Michael A. Nielsen. *Neural networks and deep learning*. `http://neuralnetworksanddeeplearning.com/chap5.html`. Jan. 1970.

[31] Dor Bank, Noam Koenigstein, and Raja Giryes. "Autoencoders". In: *arXiv preprint arXiv:2003.05991* (2020).

[32] *Using deep learning models and convolutional Neural Networks.* `https://docs.ecognition.com/eCognition_documentation/User20Guide20Developer/820Classification20-20Deep20Learning.htm`.

[33] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[34] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778.

[35] Christian Szegedy et al. "Going deeper with convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[36] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[37] Li Wan et al. "Regularization of neural networks using dropconnect". In: *International conference on machine learning*. PMLR. 2013, pp. 1058–1066.

[38] John Hertz, Anders Krogh, and Richard G Palmer. *Introduction to the theory of neural computation*. CRC Press, 2018.

[39] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[40] Sergey Ioffe and Christian Szegedy Batch Normalization. "Accelerating deep network training by reducing internal covariate shift". In: *arXiv preprint arXiv:1502.03167* (2014).

[41] Agnes Lydia and Sagayaraj Francis. "Adagrad—an optimizer for stochastic gradient descent". In: *Int. J. Inf. Comput. Sci* 6.5 (2019), pp. 566–568.

[42] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *International conference on machine learning*. PMLR. 2015, pp. 448–456.

[43] Shibani Santurkar et al. "How does batch normalization help optimization?" In: *Advances in neural information processing systems* 31 (2018).

[44] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. "Sequential model-based optimization for general algorithm configuration". In: *International conference on learning and intelligent optimization*. Springer. 2011, pp. 507–523.

[45] James Bergstra et al. "Algorithms for hyper-parameter optimization". In: *Advances in neural information processing systems* 24 (2011).

[46] George De Ath et al. "Greed is good: Exploration and exploitation trade-offs in Bayesian optimisation". In: *ACM Transactions on Evolutionary Learning and Optimization* 1.1 (2021), pp. 1–22.

[47] Han Xu et al. "Sequence determinants of improved CRISPR sgRNA design". In: *Genome research* 25.8 (2015), pp. 1147–1157.

[48] Nathan Wong, Weijun Liu, and Xiaowei Wang. "WU-CRISPR: characteristics of functional guide RNAs for the CRISPR/Cas9 system". In: *Genome biology* 16.1 (2015), pp. 1–8.

[49] Laurence OW Wilson et al. "High activity target-site identification using phenotypic independent CRISPR-Cas9 core functionality". In: *The CRISPR Journal* 1.2 (2018), pp. 182–190.

[50] Achal Rastogi et al. "PhytoCRISP-Ex: a web-based and stand-alone application to find specific target sequences for CRISPR/CAS editing". In: *BMC bioinformatics* 17.1 (2016), pp. 1–4.

[51] Aaron McKenna and Jay Shendure. "FlashFry: a fast and flexible tool for large-scale CRISPR target design". In: *BMC biology* 16.1 (2018), pp. 1–6.

[52] Jennifer Listgarten MicrosoftResearch. *MicrosoftResearch/Azimuth: Machine Learning-based predictive modelling of CRISPR/Cas9 guide efficiency*. 2016. URL: `https://github.com/MicrosoftResearch/Azimuth`.

[53] John G Doench et al. "Optimized sgRNA design to maximize activity and minimize off-target effects of CRISPR-Cas9". In: *Nature biotechnology* 34.2 (2016), pp. 184–191.

[54] Nahye Kim et al. "Prediction of the sequence-specific cleavage activity of Cas9 variants". In: *Nature Biotechnology* 38.11 (2020), pp. 1328–1336.

[55] Scott M Lundberg and Su-In Lee. "A unified approach to interpreting model predictions". In: *Advances in neural information processing systems* 30 (2017).

[56] Leo Breiman. "Random forests". In: *Machine learning* 45.1 (2001), pp. 5–32.

[57] Ferhat Alkan et al. "CRISPR-Cas9 off-targeting assessment with nucleic acid duplex energy parameters". In: *Genome biology* 19.1 (2018), pp. 1–13.

[58] Raj Chari et al. "Unraveling CRISPR-Cas9 genome engineering parameters via a library-on-library approach". In: *Nature methods* 12.9 (2015), pp. 823–826.

[59] Traver Hart et al. "High-resolution CRISPR screens reveal fitness genes and genotype-specific cancer liabilities". In: *Cell* 163.6 (2015), pp. 1515–1526.

[60] Hayden C Metsky et al. "Designing sensitive viral diagnostics with machine learning". In: *Nature biotechnology* (2022), pp. 1–9.