Project Report

on

Off-target predictions in CRISPR-Cas9 gene editing using deep learning

(A dissertation submitted in partial fulfillment of the requirements of Bachelor of Technology in Computer Science and Engineering of the Maulana Abul Kalam Azad University of Technology, West Bengal)

Submitted by

Anam Hayat

Under the guidance of
Dr. Sarit Chakraborty

Asst. Prof./Lecturer,

Dept. of Computer Science and Engineering

Government College of Engineering and Leather Technology
(Affiliated to MAKAUT, West Bengal)

Kolkata - 700106, WB

2020-2021

Certificate of Approval

This is to certify that the project report on Off-target predictions in CRISPR-Cas9 gene editing using deep learning is a record of bonafide work, carried out by Smt. Anam Hayat under my guidance and supervision

In my opinion, the report in its present form is in conformity as specified by Government College of Engineering and Leather Technology and as per regulations of the Maulana Abul Kalam Azad University of Technology, West Bengal. To the best of my knowledge the results presented here are original in nature and worthy of incorporation in project report for the B.Tech. Program in Computer Science and Engineering.

Dr. Sarit Chakraborty

Supervisor/ Guide

ACKNOWLEDGEMENT

With great pleasure, I would like to express my profound gratitude and indebtedness to Dr. Sarit Chakraborty, Department of Computer Science and Engineering, Government College of Engineering and Leather Technology, W.B. for his continuous guidance, valuable advice, and constant encouragement throughout the project work. His valuable and constructive suggestions at many difficult situations are immensely acknowledged. I am in short of words to express his contribution to this thesis through criticism, suggestions, and discussions.

I would like to take this opportunity to thank Dr. Surjadeep Sarkar, Project Coordinator and Prof. Santanu Halder, HOD, Department of Computer Science & Engineering and Information Technology, Government College of Engineering and Leather Technology.

I would like to express my gratitude to Mr. SubinayAdhikary for their valuable suggestions and help.

1. Anam Hayat - 11200118017 Signature: Anam Hayat

ABSTRACT

The CRISPR (clustered regularly interspaced short palindromic repeat)–Cas9 (CRISPR-associated nuclease 9) gene editing tool system is going to transform the developmental biology rendering a simple and efficient method to modify and regulate the genome with targeted precision and efficiency has never been easier in mammalian systems. Gene therapy with CRISPR–Cas is a potential therapeutic strategy for treating genetic neurological diseases, including Huntington disease, the spinocerebellar ataxias, fragile X syndrome, and amyotrophic lateral sclerosis. The tremendous progress with CRISPR–Cas in vitro—and, more recently, in vivo—holds great promise for future clinical translation. Here, we describe CRISPR–Cas gene editing mechanisms and the engineering advances that are optimizing this technology. We then focus on its application and therapeutic potential in preclinical studies of models of neurological disease.

The prediction of off-target mutations in CRISPR-Cas9 is a hot topic due to its relevance to gene editing research. Existing prediction methods have been developed; however, most of them just calculated scores based on mismatches to the guide sequence in CRISPR-Cas9. Therefore, the existing prediction methods are unable to scale and improve their performance with rapid expansion of experimental data in CRISPR-Cas9. Moreover, the existing methods still cannot satisfy enough precision in off-target predictions for gene editing at the clinical level.

CONTENTS

INTRODUCTION

1.1 Motivation

CRISPR technology is simple yet powerful tool for genome editing. It allows researchers to alter DNA sequences easily and modify gene function. Its many potential applications include correcting genetic defects, treating and preventing the spread of diseases and improving crops.

CRISPR technology was adapted from the natural defense mechanisms of bacteria and archaea (domain of single-celled microorganisms). These organisms use CRISPR-derived RNA and various Cas proteins, including Cas9, to foil attacks by viruses and other foreign bodies. They do so primarily by chopping up and destroying the DNA of a foreign invader. When these components are transferred into other, more complex, organisms, it allows for the manipulation of genes, or “editing”.

“CRISPR” stands for “clusters of regularly interspaced short palindromic repeats”. It is a specialized region of DNA with two distinct characteristics: the presence of nucleotide repeats and spacers. Repeated sequences of nucleotides- the building blocks of DNA- are distributed throughout a CRISPR region. Spacers are bits of DNA that are interspaced among these repeated sequences.
In case of bacteria, the spacers are taken from viruses that previously attacked the organism. They serve as a bank of memories, which enable bacteria to recognize the viruses and fight off future attacks.

CRISPR RNA (crRNA): Once a spacer is incorporated and the virus attacks again, a portion of the CRISPR is transcribed and processed into CRISPR RNA, or “crRNA”. The nucleotide sequence of the CRISPR acts as a template to produce a complementary sequence of single-stranded RNA. Each crRNA consists of nucleotide repeat and a spacer portion, according to a 2014 review by Jennifer Doudna and Emmanuelle Charpentier, published in the journal Science.

Cas9: The Cas9 protein is an enzyme that cuts foreign DNA.
The protein typically binds two RNA molecules: crRNA and another called tracrRNA(or “trans-activating crRNA”). The two then guide Cas9 to the target site where it will make its cut. This expanse of DNA is complementary to a 20 nucleotide stretch of the crRNA.
Using two separate regions, or “domains” on its structure, Cas9 cuts both strands of the DNA double helix, making what is known as a “double -stranded break”, according to the 2014 Science article.
There is a built-in safety mechanism, which ensures that Cas9 doesn't just cut anywhere in a genome. Short DNA sequences known as PAMs ("protospacer adjacent motifs") serve as tags and sit adjacent to the target DNA sequence. If the Cas9 complex doesn't see a PAM next to its target DNA sequence, it won't cut. This is one possible reason that Cas9 doesn't ever attack the CRISPR region in bacteria, according to a 2014 review published in Nature Biotechnology.

CRISPR-Cas9 as a genome-editing tool

The genomes of various organisms encode a series of messages and instructions within their DNA sequences. Genome editing involves changing those sequences, thereby changing the messages. This can be done by inserting a cut or break in the DNA and tricking a cell's natural DNA repair mechanisms into introducing the changes one wants. CRISPR-Cas9 provides a means to do so.

1.2 Background

CRISPR/Cas9 system, as the third-generation genome editing technology, has been widely applied in target gene repair and gene expression regulation. Selection of appropriate sgRNA can improve the on-target knockout efficacy of CRISPR/Cas9 system with high sensitivity and specificity. However, when CRISPR/Cas9 system is operating, unexpected cleavage may occur at some sites, known as off-target. Presently, a number of prediction methods have been developed to predict the off-target propensity of sgRNA at specific DNA fragments. Most of them use artificial feature extraction operations and machine learning techniques to obtain off-target scores. With the rapid expansion of off-target data and the rapid development of deep learning theory, the existing prediction methods can no longer satisfy the prediction accuracy at the clinical level.

1.4 Summary of Present Work

To address the problems in the existing prediction methods, which are unable to scale and improve their performance with rapid expansion of experimental data in CRISPR-Cas9. Moreover, the existing methods still cannot satisfy enough precision in off-target predictions for gene editing at the clinical level. We have train and tested the dataset for faster and accurate prediction of off target mutation.We have also designed an algorithm using deep neural networks to predict off-target mutations.

1.5 Organization of the thesis

To predict off target mutations in CRISPR Cas9 gene editing.

1.6 Hardware/Software Used

- Google Colab
- Jupyter Notebook

2.1 CONVOLUTIONAL NEURAL NETWORK(CNN)

The name "convolutional neural network" indicates that the network employs a mathematical operation called convolution. Convolutional networks are a specialized type of neural networks that use convolution in place of general matrix multiplication in at least one of their layers. A convolutional neural network consists of an input layer, hidden layers and an output layer. In any feed-forward neural network, any middle layers are called hidden because their inputs and outputs are masked by the activation function and final convolution. In a convolutional neural network, the hidden layers include layers that perform convolutions. Typically, this includes a layer that performs a dot product of the convolution kernel with the layer's input matrix. This product is usually the Frobenius inner product, and its activation function is commonly ReLU. As the convolution kernel slides along the input matrix for the layer, the convolution operation generates a feature map, which in turn contributes to the input of the next layer. This is followed by other layers such as pooling layers, fully connected layers, and normalization layers.

2.2) ACTIVATION FUNCTION

An activation function in a neural network defines how the weighted sum of the input is transformed into an output from a node or nodes in a layer of the network.
Sometimes the activation function is called a “transfer function.” If the output range of the activation function is limited, then it may be called a “squashing function.” Many activation functions are nonlinear and may be referred to as the “nonlinearity” in the layer or the network design.
The choice of activation function has a large impact on the capability and performance of the neural network, and different activation functions may be used in different parts of the model.
Technically, the activation function is used within or after the internal processing of each node in the network, although networks are designed to use the same activation function for all nodes in a layer.
A network may have three types of layers: input layers that take raw input from the domain, hidden layers that take input from another layer and pass output to another layer, and output layers that make a prediction.
All hidden layers typically use the same activation function. The output layer will typically use a different activation function from the hidden layers and is dependent upon the type of prediction required by the model.
Activation functions are also typically differentiable, meaning the first-order derivative can be calculated for a given input value. This is required given that neural networks are typically trained using the backpropagation of error algorithm that requires the derivative of prediction error in order to update the weights of the model.
There are many different types of activation functions used in neural networks, although perhaps only a small number of functions used in practice for hidden and output layers.

2.1) MATERIALS AND METHODS

2.1.1) SEQUENCE ENCODING

For encoding, the complementary base is designed to represent the original base in sgRNA; for instance, we can use A, G, C, T to represent both sgRNA and target DNA sequence in CRISPR-Cas9. Therefore, each base in the sgRNA and target DNA can be encoded as one of the four one-hot vectors [1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0] and [0, 0, 0, 1]. As a result, every sgRNA-DNA sequence pair can be represented by a 4 x 23 matrix where 23 is the length of the sequence which includes the 3-bp PAM adjacent to the 20 bases. To encode the mutated information in sgRNA-DNA, we derived a 4-length vector to encode the mismatched base pairs by implementing OR operator on two one-hot vectors of base-pairing. The code matrix of sgRNA DNA will be directly fed into CNN-based models for training and testing, while the vectorization of the encoding matrix will be used as the input of traditional machine-learning-based models and deep FNN.

2.1.2) NEURAL NETWORK MODELS

The first layer of our network is a convolutional layer, which is designed for extracting sgRNA-DNA matching information using 40 filters of different sizes (10 for each of the sizes 4 x 1, 4 x 2, 4 x 3 and 4 x 5). To preserve the integrity of every base pair code in gRNA-DNA, the size of scanning step for each filter is set to 4 in the dimension of base pair. Thus, it gives a 1 x 23 x 40 feature map from this layer. The second layer is a batch normalization (BN) layer, which is designed for reducing internal covariate shift in the neural network (Sergey and Christian, 2015). It further prevents smaller changes to the parameters to amplify and thereby allows higher learning rates than the opposite case. Moreover, ReLU is used as the activation function for each neuron in this layer. The third layer is a global max-pooling layer connected with the previous BN layer. Each of these max-pooling windows only outputs the maximum value of all of its respective BN layer outputs. The size of each pooling window used in standard CNN (CNN_std) is 1 x5; other sizes are also tested in the following experiments. Accordingly, it gives a 1 x 5 x 40 feature map to the next layers. The function of this global max-pooling can be thought as calling whether the mismatches modelled by the respective BN layer exist in the input sequence or not. The following layers are two fully connected dense layers with the sizes of 100 and 23, respectively. A dropout layer is used on the last dense layer to randomly mask portions of the output to avoid overfitting; the probability used in CNN_std to drop a unit is 0.15 (Srivastava et al., 2014). The final output layer consists of two neurons corresponding to the two classification results. Those two neurons are fully connected to the previous layers. The architecture of FNN used for off-target prediction consists of input layer, several hidden layers and output layer. The input of FNN model is a vector with the length of 92, the vectorization of 4 x 23 matrix. The activation function is softmax which is able to convert each neuron output into probability. Accordingly cross entropy (Lih-Yuan, 2006) is chosen as the loss function for our FNN.

2.2) CURRENT PREDICTION ALGORITHMS

There are four most recent off-target prediction algorithms including CFD score, MIT score, CROP-IT score and CCTop score. In order to study whether the potential off target is validated or not, these methods calculate scores based on the positions or identities of the mismatches to the guide RNA sequence. Higher score means this sequence is more likely to be an active off-target than the control. Initially, systematic testing of the effect of mismatches led to a weight for each possible nucleotide change at each position.

2.3) MODEL SELECTION

We compare the performance of different model architectures trained on the CRISPOR dataset. FNN_3layer and CNN_std achieved the best performance predicting off-targets under stratified 5-fold cross-validation with the mean AUCs of 0.970 and 0.972, respectively. Based on the validation results of the CNN models (i.e., CNN_std, CNN_np, CNN_pool_win3 and CNN_pool_win3) used for pooling layer testing, we found that pooling layer significantly increased performance for off-targets prediction. In particular, the CNN with pooling layer of window size 5 (CNN_std) achieved the best performance; such observation emphasizes the need to use a pooling layer with befitting window size to extract the mismatches in sgRNA-DNA. Comparing CNN_std with CNN_nbn, we see that BN improves the performance. This is expected since BN can stabilize the training process and decrease the risk of overfitting. The performance of CNN_nd comparing with the CNN_std shows that drop-out layer can slightly improve the generalization performance. Comparing FNN with CNN, we found that the average Mean AUC of CNN models are higher than that of FNN models while FNNs had lower average standard deviance.

2.3) COMPARISION TO CURRENT ALGORITHM

The performance of FNN_3layer and CNN_std were compared with the current off-targets prediction models: CFD score, MIT score, MIT web score, CROP-IT score and CCTop score under stratified 5-fold cross-validation on CRISPOR dataset. It is hardly surprising that CFD score achieved the best performance with AUC of 0.912 among all current off-targets prediction models, since the experiments from Haeussler et al. have already proved that CFD score was the best prediction model on CRISPOR dataset. However, our two deep learning models, CNN_std and FNN_3layer, achieved much better performance than all current off-targets prediction models in both ROC curves and AUC values. The AUCs of our CNN and FNN models are roughly 5.8% higher than CFD score under stratified 5-fold cross-validation on CRISPOR dataset, reaching 0.972 and 0.970, respectively. Furthermore, the ROC curves of both deep learning models completely covered the ROC curve of CFD score. These results reveal that our deep learning approaches have competitive edges over the existing methods on the CRISPOR dataset.

2.4) COMPARISION TO TRADITIONAL MACHINE LEARNING MODEL

Since the current off-target prediction models rely on the fixed scores to represent the mismatched information to evaluate the potential off-target sites, they do have the ability to improve their performance by training. Therefore, we decided to implement some traditional machine-learning models including LR, RF, and GBT for further comparison. Three traditional and two deep learning models were trained and tested on CRISPOR dataset under stratified 5-fold cross-validation. The standard deviations of two deep learning models are the lowest among all models; it reveals that our deep learning models are more stable than the traditional machine learning methods and current prediction models on the CRISPOR dataset. In addition, we found that LR and GBT achieved slightly better performance with AUCs of 0.931 and 0.914 than CFD score. The observations confirm that machine-learning-based method still have good potential in off-target predictions for CRISPR-Cas9 gene editing.

3. ENVIRONMENT AND LIBRARIES

3.1 ENVIRONMENT

We had chosen Google Colaboratory as the environment for the complete development of our model. There are other alternatives but we had opted for Google Colaboratory due to the ease of access and on cloud services it provides.

3.1.1 GOOGLE COLABORATORY

Colab is a free Jupyter notebook environment that runs entirely in the cloud. Most importantly, it does not require a setup and the notebooks that you create can be simultaneously edited by your team members - just the way you edit documents in Google Docs. Colab supports many popular machine learning libraries which can be easily loaded in your notebook.

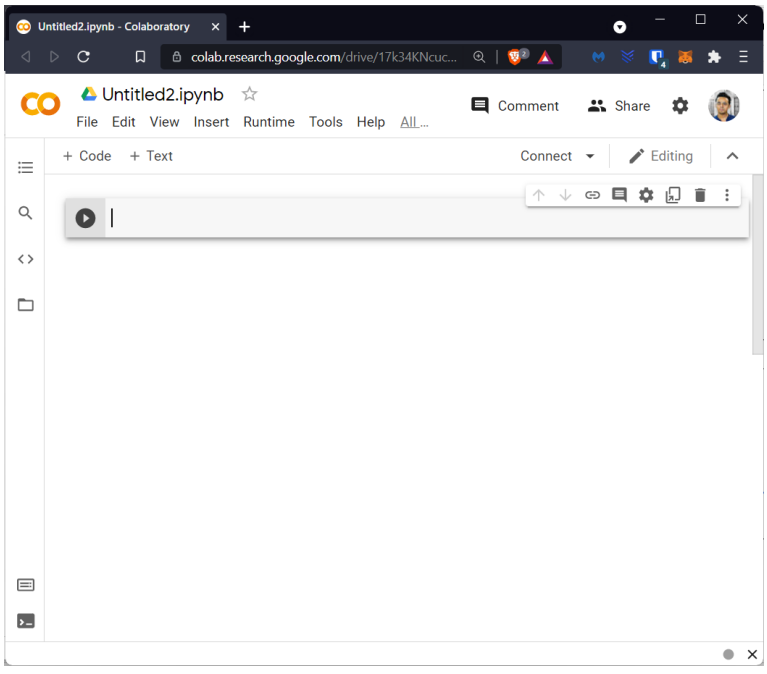


Fig. 3.1: Colab notebooks allow you to combine executable code and rich text in a single document, along with images, HTML, LaTeX and more.

3.1.2 FEATURES OF COLAB

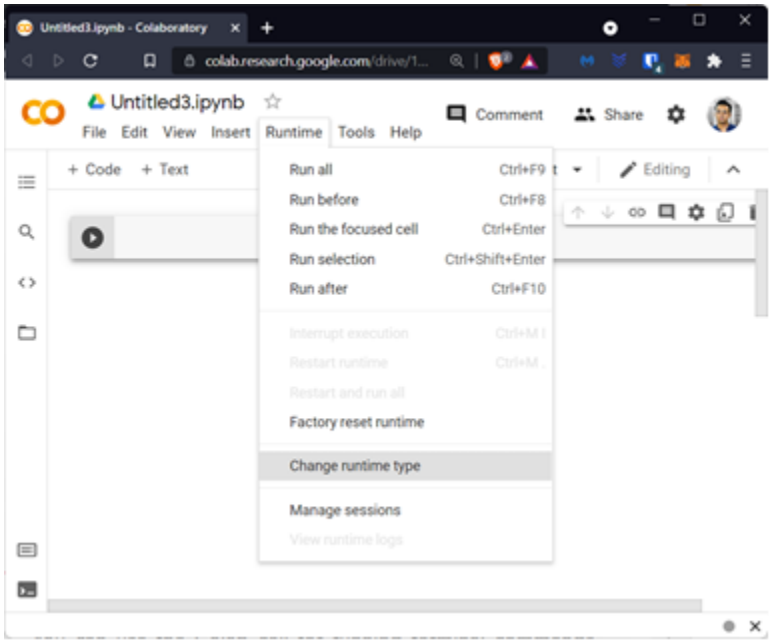
Colab offers us different set of tools to make our life easier. Features are such as:

• Python is used for writing and executing code.
• Mathematical equations can be easily given as input and can be processed.
• Create/Upload/Share notebooks
• Import/Save notebooks from/to Google drive.
• Import external datasets e.g. from Kaggle
• Integrate PyTorch, Tensorflow.
• Free cloud service with free GPU.

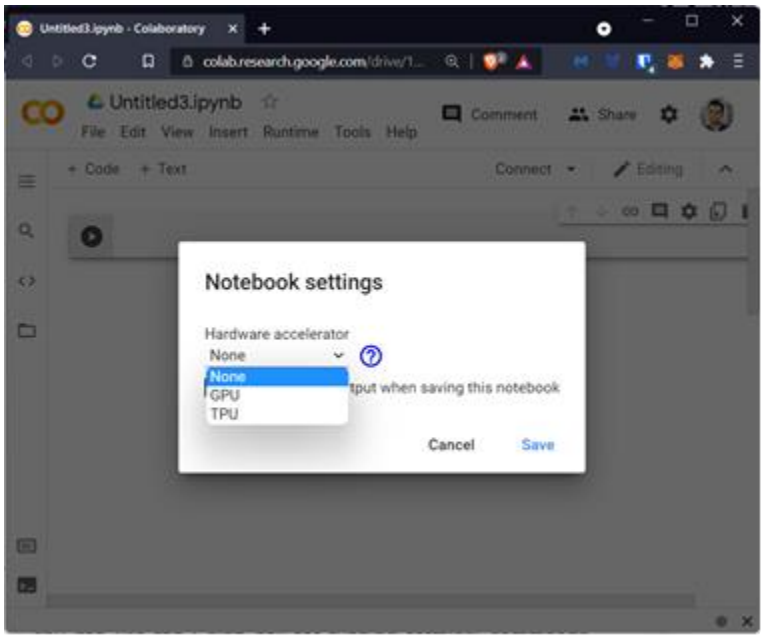
3.1.3 GOOGLE COLAB RUNTIMES – CHOOSING THE GPU OR TPU OPTION

The ability to choose different types of runtimes is what makes Colab so popular and powerful. Here are the steps to change the runtime of your notebook:

Step 1: Click ‘Runtime’ on the top menu and select ‘Change Runtime Type’:



Step 2: Here you can change the runtime according to your needs.



3.1.4 USING TERMINAL COMMANDS ON GOOGLE COLAB

You can use the Colab cell for running terminal commands. Most of the popular libraries come installed by default on Google Colab. Python libraries like Pandas, NumPy, scikitlearn are all pre-installed. If you want to run a different Python library, you can always install it inside your Colab notebook like this: !pip install library_name Everything is similar to how it works in a regular terminal. We just you have to put an exclamation(!) before writing each command like: !ls

3.1.5 GPUS AND TPUS ON GOOGLE COLAB

Training models, especially deep learning ones, takes numerous hours on a CPU. We've all faced this issue on our local machines. GPUs and TPUs, on the other hand, can train these models in a matter of minutes or seconds.

It gives you a decent GPU for free, which you can continuously run for 12 hours. For most data science folks, this is sufficient to meet their computation needs.

Google Colab gives us three types of runtimes for our notebooks:

• CPUs,
• GPUs, and
• TPUs

As mentioned, Colab gives us 12 hours of continuous execution time. After that, the whole virtual machine is cleared and we have to start again. We can run multiple CPU, GPU, and TPU instances simultaneously, but our resources are shared between these instances.

3.2 LIBRARIES

We have used the following libraries:

```
from numpy import array
from numpy import argmax
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
import re
```

```
import pandas as pd
import numpy as np
import io
from google.colab import files
uploaded=files.upload()
```

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import layers
```

3.2.1 NUMPY

NumPy (short for Numerical Python) is “the fundamental package for scientific computing with Python” and it is the library Pandas, Matplotlib and Scikit-learn builds on top off.

Functions: -

- Creating NumPy Arrays, Loading and Saving Files
- Working and Inspecting Arrays
- Indexing and Slicing
- Sorting and Reshaping
- Combining and Splitting
- Adding and Removing Elements
- Descriptive Statistics

3.2.2 SKLEARN

Scikit-learn (formerly scikits.learn and also known as sklearn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

sklearn.preprocessing.LabelEncoder - Encode target labels with value between 0 and n_classes-1.

sklearn.preprocessing.OneHotEncoder - Encode categorical features as a one-hot numeric array.

The input to this transformer should be an array-like of integers or strings, denoting the values taken on by categorical (discrete) features. The features are encoded using a one-hot (aka 'one-of-K' or 'dummy') encoding scheme. This creates a binary column for each category and returns a sparse matrix or dense array (depending on the sparse parameter)

3.2.3 RE (Regular Expression)

Python has a module named re to work with RegEx. A Regular Expression (RegEx) is a sequence of characters that defines a search pattern. For example, ^a...s$

3.2.4 PANDAS

Pandas is quite a game changer when it comes to analyzing data with Python and it is one of the most preferred and widely used tools in data munging/wrangling if not THE most used one. Pandas is an open source, free to use (under a BSD license) and it was originally written by Wes McKinney.

Pandas takes data (like a CSV or TSV file, or a SQL database) and creates a Python object with rows and columns called data 15 frame that looks very similar to table in a statistical software (think Excel or SPSS for example. This is so much easier to work with in comparison to working with lists and/or dictionaries
through for loops or list comprehension.

Functions: -

• Loading and Saving Data
• Viewing and Inspecting Data
• Selection of Data
• Filter, Sort and Groupby
• Data Cleaning
• Join/Combine

Pandas provides a nice utility function "json_normalize" for flattening semi-structured JSON objects.

3.2.4 PYTORCH

PyTorch is a Python package that provides two high-level features:

- Tensor computation (like NumPy) with strong GPU acceleration
- Deep neural networks built on a tape-based autograd system
-

We can reuse your favorite Python packages such as NumPy, SciPy, and Cython to extend PyTorch when needed.

Usually, PyTorch is used either as:

- A replacement for NumPy to use the power of GPUs.
- A deep learning research platform that provides maximum flexibility and speed.

PyTorch has a unique way of building neural networks: using and replaying a tape recorder.

Most frameworks such as TensorFlow, Theano, Caffe, and CNTK have a static view of the world. One has to build a neural network and reuse the same structure again and again. Changing the way, the network behaves means that one has to start from scratch.

With PyTorch, we use a technique called reverse-mode auto-differentiation, which allows you to change the way your network behaves arbitrarily with zero lag or overhead. Our inspiration comes from several research papers on this topic, as well as current and past work such as torch-autograd, autograd, Chainer, etc.

While this technique is not unique to PyTorch, it's one of the fastest implementations of it to date. You get the best of speed and flexibility for your crazy research.

3.2.5 MATPLOTLIB

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.

Matplotlib is one of the most powerful tools for data visualization in Python. It tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, error-charts, scatterplots, etc., with just a few lines of code.

3.2.6 TENSERFLOW

TensorFlow is an open-source software library for high performance numerical computation. Its flexible architecture allows easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices.
Originally developed by researchers and engineers from the Google Brain team within Google's AI organization, it comes with strong support for machine learning and deep learning and the flexible numerical computation core is used across many other scientificdomains.

4. Code Implementation

4.1 OBJECTIVE

The prediction of off-target mutations in CRISPR-Cas9 is a hot topic due to its relevance to gene editing research. Existing prediction methods have been developed; however, most of them just calculated scores based on mismatches to the guide sequence in CRISPR-Cas9. To address it, we design an algorithm using deep neural networks to predict off-target mutations in CRISPR-Cas9 gene editing.

4.2 LIBRARY USED

- Numpy
- Pandas
- Sklearn (LabelEncoder&OneHotEncoder)
- Re
- Tensorflow(keras)
- Torchvision

4.2 Uploading Datasets

```
import pandas as pd
import numpy as np
import io
from google.colab import files
uploaded=files.upload()
```

```
Choose Files No file chosen        Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving sequence.xlsx to sequence.xlsx
```

4.3 Reading Datasets

Using pd.read_excel(io.BytesID[]) command we read the dataset and print all the dataset to take a glance at the different attributes present in it.

```
df=pd.read_excel(io.BytesIO(uploaded['sequence.xlsx']))
df
```

	guideId	guideSeq	offtargetSeq	mismatchPos	mismatchCount	mitOfftargetScore	cfdOfftargetScore	chrom	start
0	46rev	TTCCGGCGCGCCGAGTCCTTAGG	TCCCAGTGCTCCGAGTCCTTCGG	.*.*.*..*.....	4	0.858608	0.490402	chr14	70883813
1	46rev	TTCCGGCGCGCCGAGTCCTTAGG	TTCCAGCAGGCAGAGTCCTTGGG	....*..**..*.....	4	0.410318	0.383220	chr2	120245677
2	46rev	TTCCGGCGCGCCGAGTCCTTAGG	TTCCGGTGTGCACAGTCCTTTGG	.....*.*..**.....	4	0.108456	0.213816	chr1	207844276
3	46rev	TTCCGGCGCGCCGAGTCCTTAGG	TTTGGCTGCGCCGAGTCCTTTGG	..**.**.....	4	0.531674	0.190430	chr22	49591793
4	46rev	TTCCGGCGCGCCGAGTCCTTAGG	TACCTGCGCGCCGAGAACTTCGG	.*..*.....**...	4		0.107692	chr17	76587006
...	...	...	...	...	...	...	...	...	...
7359	232forw	GCCGTCTTCCCCTCCATCGTGGG	GCCGTCTTCCCCTCCATCGTGGG	.....	0	100	1.000000	chr11	1825219
7360	213rev	CCCCACGATGGAGGGGAAGACGG	CCCCACGATGGAGGGGAAGACGG	.....	0	100	1.000000	chr11	1825218
7361	213rev	CCCCACGATGGAGGGGAAGACGG	CCCCACGATGGAGGGGAAGATGG	.....	0	100	1.000000	chr19	9628911
7362	184forw	GGCTCCGGCATGTGCAAGGCCGG	GGCTCCGGCATGTGCAAGGCCAG	.....	0	20	0.259259	chr5	77080747
7363	180forw	CAACGGCTCCGGCATGTGCAAGG	CAACGGCTCCGGCATGTGCAAGG	.....	0	100	1.000000	chr5	77080743

7364 rows × 12 columns

4.4 Finding description of different attribute

```
df['guideSeq'].describe()
```

```
count                          7364
unique                           52
top       GGGCGCGCGCCGGCGCCCCCTGG
freq                            429
Name: guideSeq, dtype: object
```

4.5 Matrix representation of DNA sequence using Onehot Encoding

```
from numpy import array
from numpy import argmax
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
import re
```

```
class hot_dna:
 def __init__(self,fasta):

  #check for and grab sequence name
  if re.search(">",fasta):
   name = re.split("\n",fasta)[0]
   sequence = re.split("\n",fasta)[1]
  else :
   name = 'unknown_sequence'
   sequence = fasta

  #get sequence into an array
  seq_array = array(list(sequence))

  #integer encode the sequence
  label_encoder = LabelEncoder()
  integer_encoded_seq = label_encoder.fit_transform(seq_array)
```

```
  #one hot the sequence
  onehot_encoder = OneHotEncoder(sparse=False)
  #reshape because that's what OneHotEncoder likes
  integer_encoded_seq = integer_encoded_seq.reshape(len(integer_encoded_seq), 1)
  onehot_encoded_seq = onehot_encoder.fit_transform(integer_encoded_seq)

  #add the attributes to self
  self.name = name
  self.sequence = fasta
  self.integer = integer_encoded_seq
  self.onehot = onehot_encoded_seq
```

4.6.1 Implementing One-Hot code

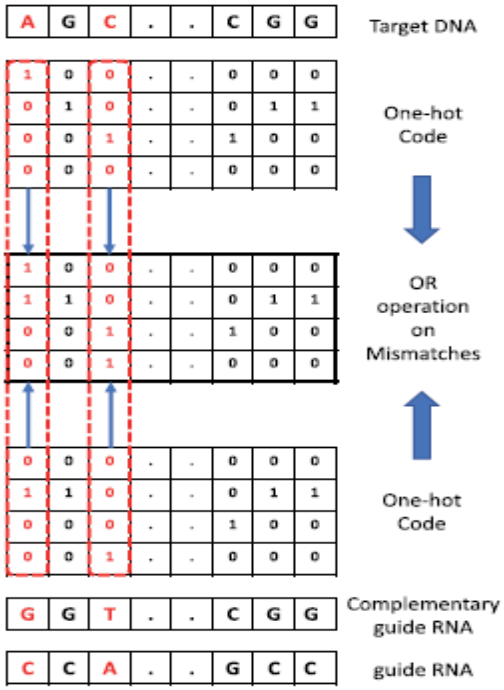
```
my_code1 = hot_dna(df['guideSeq'][0])
my_code2=hot_dna(df['offtargetSeq'][0])
arr2=np.array(my_code2.onehot)
arr1=np.array(my_code1.onehot)
arr3=arr1+arr2
for k in range(23):
  for j in range(4):
    if arr3[k][j]>0:
      arr3[k][j]=1
print(arr3)
```

```
temp_list=[]
for i in range (32):
  my_code1 = hot_dna(df['guideSeq'][i])
  my_code2=hot_dna(df['offtargetSeq'][i])

  arr2=np.array(my_code2.onehot)
  arr1=np.array(my_code1.onehot)
  arr32=arr1+arr2
  for k in range(23):
    for j in range(4):
      if arr32[k][j]>0:
        arr32[k][j]=1
  x=arr32.size
  print(arr32)

  DF=pd.DataFrame(arr32)
  temp_list.append(DF)
  print(len(temp_list))
newdf=pd.concat(temp_list)
newdf.to_csv("Data.csv")
```

4.6.1 Explanation



4.7 Storing matrices into “Data.csv” File

Diving dataset into training and testing dataset

```
abalone_train=pd.read_csv("Data.csv")
```

```
[ ] abalone_features=abalone_train[1:31]
    abalone_features_test=abalone_train[31:33]
    abalone_lables=abalone_features_test.pop('1')
    abalone_lables=abalone_features.pop('1')
```

4.8 Training Neural network model

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import layers
```

```
[ ] abalone_model=tf.keras.Sequential([layers.Dense(64),
                                       layers.Dense(1)])
    abalone_model.compile(loss=tf.losses.MeanSquaredError(),optimizer=tf.optimizers.Adam(),metrics=[tf.keras.metrics.Accuracy()])
```

```
[ ] abalone_model.fit(abalone_features,abalone_lables,epochs=10)
```

4.9 Output

```
Epoch 1/10
1/1 [==============================] - 0s 381ms/step - loss: 0.6342 - accuracy: 0.0000e+00
Epoch 2/10
1/1 [==============================] - 0s 5ms/step - loss: 0.4563 - accuracy: 0.0000e+00
Epoch 3/10
1/1 [==============================] - 0s 6ms/step - loss: 0.3954 - accuracy: 0.0000e+00
Epoch 4/10
1/1 [==============================] - 0s 5ms/step - loss: 0.4159 - accuracy: 0.0000e+00
Epoch 5/10
1/1 [==============================] - 0s 7ms/step - loss: 0.4518 - accuracy: 0.0000e+00
Epoch 6/10
1/1 [==============================] - 0s 4ms/step - loss: 0.4592 - accuracy: 0.0000e+00
Epoch 7/10
1/1 [==============================] - 0s 4ms/step - loss: 0.4374 - accuracy: 0.0000e+00
Epoch 8/10
1/1 [==============================] - 0s 4ms/step - loss: 0.4039 - accuracy: 0.0000e+00
Epoch 9/10
1/1 [==============================] - 0s 5ms/step - loss: 0.3755 - accuracy: 0.0000e+00
Epoch 10/10
1/1 [==============================] - 0s 7ms/step - loss: 0.3614 - accuracy: 0.0000e+00
<tensorflow.python.keras.callbacks.History at 0x7fc0bfcae610>
```

CONCLUSIONS

We presented that deep neural networks can accurately predict the off-targets of CRISPR-Cas9 gene editing. To our knowledge, this is the first time that deep neural networks are designed and implemented for off-target predictions. Our final CNN, CNN_std, obtained the best performance on both CRISPOR dataset and GUIDE-seq dataset, outperforming the current state-of-art off-target prediction methods and three traditional machine learning algorithms including LR, RF and GBT. We discussed and attributed its performance successes to the neural network layer designs which are general enough to self-learn and capture sequence features. We believe that such intelligent approaches can contribute to CRISPR-Cas9 off-target predictions or other similar problems in a rigorous manner.

REFERENCES

[1]Jiecong Lin and Ka-Chun Wong (September 2018) – Off Target Predictions in CRISPR -Cas9 gene editing using deep learning

[2]https://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-020-3395-z

[3]https://www.livescience.com/58790-crispr-explained.html

[4]https://pubmed.ncbi.nlm.nih.gov/29446747/

[5]https://www.sciencedirect.com/topics/neuroscience/crispr

[6]https://www.livescience.com/58790-crispr-explained.html

[7]https://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-020-3395-z