**Prediction of DNA-Protein Interaction using CNN and LSTM.**

Development of a deep neural network to predict whether a DNA sequence can bind to a protein or not, explained with code.

[[](https://medium.com/@mkk.rakesh?source=post_page-----0a7aa7e074f1--------------------------------)](https://medium.com/@mkk.rakesh?source=post_page-----0a7aa7e074f1--------------------------------)

[[Towards AI](https://pub.towardsai.net/?source=post_page-----0a7aa7e074f1--------------------------------)](https://pub.towardsai.net/?source=post_page-----0a7aa7e074f1--------------------------------)

[Rakesh M K](https://medium.com/@mkk.rakesh?source=post_page-----0a7aa7e074f1--------------------------------)

·

[Follow](https://medium.com/m/signin?actionUrl=https%3A%2F%2Fmedium.com%2F_%2Fsubscribe%2Fuser%2F5796472804b9&operation=register&redirect=https%3A%2F%2Fpub.towardsai.net%2Fprediction-of-dna-protein-interaction-using-cnn-and-lstm-0a7aa7e074f1&user=Rakesh+M+K&userId=5796472804b9&source=post_page-5796472804b9----0a7aa7e074f1---------------------post_header-----------)

Published in

[Towards AI](https://pub.towardsai.net/?source=post_page-----0a7aa7e074f1--------------------------------)

·

5 min read

·

5 days ago

347



Photo by [National Cancer Institute](https://unsplash.com/@nci?utm_source=medium&utm_medium=referral) on [Unsplash](https://unsplash.com/?utm_source=medium&utm_medium=referral)

**Table of contents**

1. Introduction

2. About Data

3. Data Preprocessing

4. Model Building

5. Performance Analysis

6. Conclusion

7. References

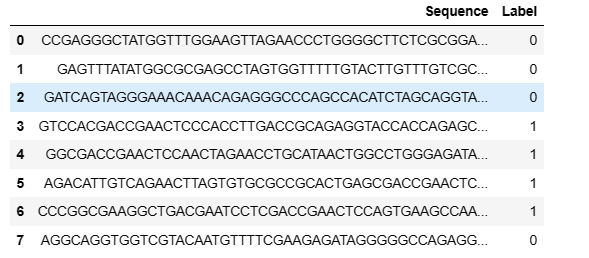
**Introduction**

To begin with, let’s keep in mind that there exists some binding between DNA and protein. DNA-binding proteins are proteins that have DNA-binding domains and thus have a specific or general affinity for single- or double-stranded DNA [1]. Keeping genomics apart, the aim of my work is to develop a deep neural network that can predict whether a DNA sequence can bind to a protein or not. So, we will be heading to a binary classification problem, and I intend to show how a deep neural network composed of CNN and LSTM can effectively solve it.

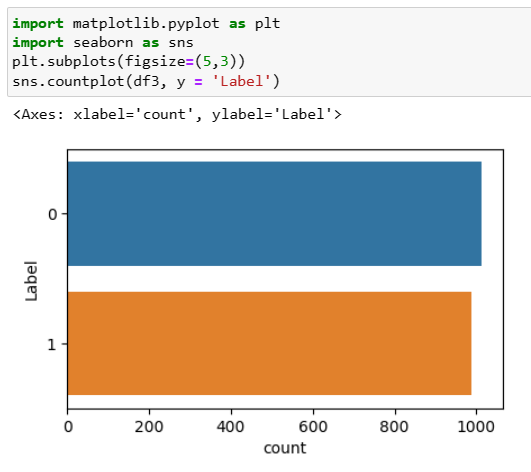
**About Data**

The DNA sequence and labels are downloaded from <https://github.com/abidlabs/deep-learning-genomics-primer/blob/master/sequences.txt> and <https://github.com/abidlabs/deep-learning-genomics-primer/blob/master/labels.txt> respectively which are publicly accessible. Let’s import and have a look at the data.

import pandas as pd  
  
df1 = pd.read\_csv('sequences.txt', sep=' ', header=None, names=[ 'Sequence'])  
df2 = pd.read\_csv('labels.txt', sep=' ', header=None, names=['Label'])  
df3 = pd.concat([df1,df2],axis=1)  
df3.head(8)



DNA consists of four bases, which are adenine [A], cytosine [C], guanine [G], or thymine [T]. DNA sequence is a laboratory process of determining the sequence of these four bases in a DNA molecule [1]. There are 2000 records, and each are of length 200. Also, the data is reasonably balanced as you see below.



**Data Preprocessing**

The text sequence needs to be converted into numbers before feeding to the model. One-hot encoding is done on the sequence since neural networks often works well with the same. The encoding code is done as below.

def one\_hot\_encode(sequence):  
 nucleotide\_to\_index = {'A': 0, 'T': 1, 'G': 2, 'C': 3}  
 one\_hot\_encoded = []  
  
 for nucleotide in sequence:  
 one\_hot\_vector = [0] \* 4  
 one\_hot\_vector[nucleotide\_to\_index[nucleotide]] = 1  
 one\_hot\_encoded.append(one\_hot\_vector)  
  
 return np.array(one\_hot\_encoded)  
  
df3['one-hot']  
  
0 [[0, 0, 0, 1], [0, 0, 0, 1], [0, 0, 1, 0], [1,...  
1 [[0, 0, 1, 0], [1, 0, 0, 0], [0, 0, 1, 0], [0,...  
2 [[0, 0, 1, 0], [1, 0, 0, 0], [0, 1, 0, 0], [0,...  
3 [[0, 0, 1, 0], [0, 1, 0, 0], [0, 0, 0, 1], [0,...  
4 [[0, 0, 1, 0], [0, 0, 1, 0], [0, 0, 0, 1], [0,...  
 ...   
1995 [[0, 0, 1, 0], [0, 1, 0, 0], [0, 0, 0, 1], [0,...  
1996 [[0, 0, 1, 0], [0, 1, 0, 0], [0, 1, 0, 0], [0,...  
1997 [[1, 0, 0, 0], [0, 0, 0, 1], [0, 1, 0, 0], [0,...  
1998 [[0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1], [1,...  
1999 [[1, 0, 0, 0], [1, 0, 0, 0], [0, 1, 0, 0], [0,...  
Name: one-hot, Length: 2000, dtype: object

**Model Building**

A combination of Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM) layers are mainly used for the model. **Conv-1D layers are mainly used in problems that involve sequential data.** Conv1D layers are efficient in capturing patterns regardless of the position in the sequence. Since these layers have fewer parameters, they are computationally efficient and also known for their ability to capture local patterns in sequential data.

**LSTM is a Recurrent Neural Network that is designed to overcome the issue of the vanishing gradient problem.** The three gates in the cell input gate, output gate and forget gate incorporate to retain and propagate information in the sequence.

The data is split to train and test as below.

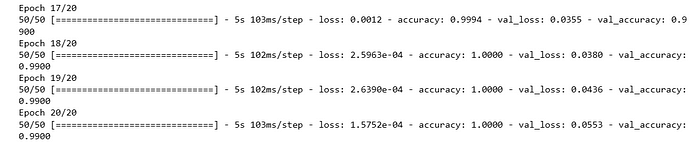
from sklearn.model\_selection import train\_test\_split  
  
'''split the data'''  
trainSeq,testSeq,trainLabels,testLabels = train\_test\_split(df3["one-hot"].to\_numpy(),  
 df3['Label'].to\_numpy(),  
 test\_size=0.2,  
 random\_state=101)  
'''convert train and test data to numpy array'''  
seqTrain1 = np.array(seqTrain)  
seqTest1 = np.array(seqTest)  
  
'''reshape the array since the neural layers expect data in 3D'''  
X\_train = np.reshape(seqTrain1, (seqTrain1.shape[0], seqTrain1.shape[1], 1)) # (1600, 200, 1)  
X\_test = np.reshape(seqTest1, (seqTest1.shape[0], seqTest1.shape[1], 1)) # (1600, 200, 1)  
  
'''check shape of train and test data'''  
X\_train.shape, X\_test.shape  
((1600, 200, 1), (400, 200, 1))

The model is configured in **TensorFlow** framework using**keras** as the below code shows.

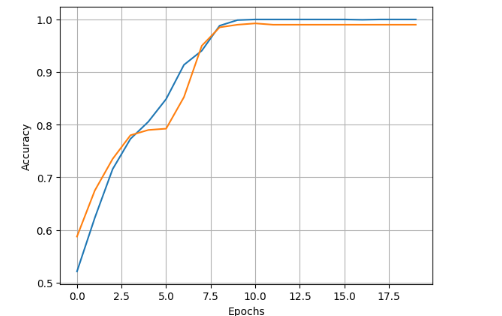
import tensorflow as tf  
from tensorflow.keras import layers  
tf.random.set\_seed(101)  
  
  
inputs= layers.Input(shape=(200,1 ))  
  
x= layers.Conv1D(filters=7,kernel\_size=5)(inputs)  
x= layers.LSTM(units=64,return\_sequences =True,)(x)  
x= layers.MaxPooling1D(2,2)(x)  
x= layers.Flatten()(x)  
x=tf.keras.layers.Masking(mask\_value=0)(x)  
x=layers.Dense(64,activation = 'relu')(x)  
x=layers.Dropout(0.1)(x)  
  
  
outputs=layers.Dense(1,activation="sigmoid")(x)  
  
modelLSTM=tf.keras.Model(inputs,outputs,name="modelLSTM")

tf.keras.layers.Masking(mask\_value=0) masks all timesteps where the value is equal to 0. During training, the masked timesteps will be ignored, and the network will not consider them, which eventually makes the computation efficient.

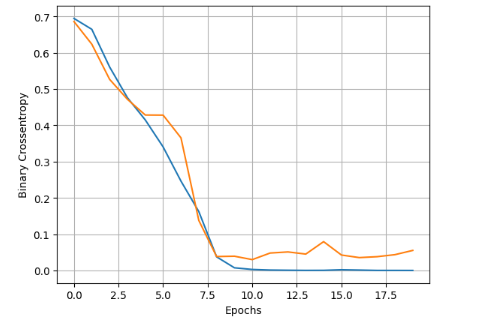
The model is trained for 20 epochs. A validation accuracy of .99 is quite impressive.



Accuracy and loss curve during training of model are shown below.



Accuracy Curve



Loss Curve

**Performance Analysis**

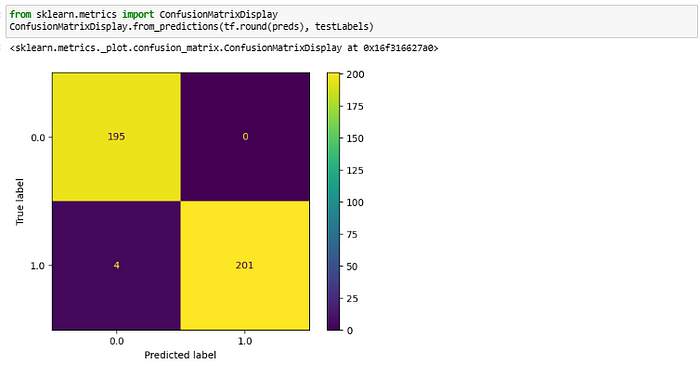
Let’s evaluate the model with test data and have a look into the classification report as well as the confusion matrix.

'''evaluating the model on test data'''  
modelLSTM.evaluate(X\_test,testLabels)  
  
[0.055308908224105835, 0.9900000095367432]

**Classification report:**

precision recall f1-score support  
  
 0.0 0.98 1.00 0.99 195  
 1.0 1.00 0.98 0.99 205  
  
 accuracy 0.99 400  
 macro avg 0.99 0.99 0.99 400  
weighted avg 0.99 0.99 0.99 400

**Confusion matrix:**



Confusion matrix

From the above results, the performance of the model is much more impressive other than negligible false negatives.

**Conclusion**

Statistical models such as SVM, KNN, and Random Forest were tried before moving into a neural network, and the maximum accuracy obtained was 0.83 with SVM. The deep model has outperformed all these models with stunning accuracy and an f1 score of 0.99.

**References**

1. [Apply Machine Learning Algorithms for Genomics Data Classification | by Ernest Bonat, Ph.D. | MLearning.ai | Medium](https://medium.com/mlearning-ai/apply-machine-learning-algorithms-for-genomics-data-classification-132972933723)
2. [Convolutional Neural Network (CNN) | TensorFlow Core](https://www.tensorflow.org/tutorials/images/cnn)