

# Journal of Visualized Experiments

## A virtual machine platform for non-computer professionals for using deep learning to classify biological sequences of metagenomic data

--Manuscript Draft--

<b>Article Type:</b>	Invited Methods Collection - Author Produced Video
<b>Manuscript Number:</b>	JoVE62250R5
<b>Full Title:</b>	A virtual machine platform for non-computer professionals for using deep learning to classify biological sequences of metagenomic data
<b>Corresponding Author:</b>	Hongwei Zhou  CHINA
<b>Corresponding Author's Institution:</b>	
<b>Corresponding Author E-Mail:</b>	hzhou@smu.edu.cn
<b>Order of Authors:</b>	Zhencheng Fang Hongwei Zhou
<b>Additional Information:</b>	
<b>Question</b>	<b>Response</b>
Please indicate whether this article will be Standard Access or Open Access.	Standard Access (US\$1200)
Please specify the section of the submitted manuscript.	Genetics
Please confirm that you have read and agree to the terms and conditions of the author license agreement that applies below:	I agree to the <a href="#">Author License Agreement</a>
Please provide any comments to the journal here.	NA
Please indicate whether this article will be Standard Access or Open Access.	Standard Access (\$1400)
Please confirm that you have read and agree to the terms and conditions of the video release that applies below:	I agree to the <a href="#">Video Release</a>

**TITLE:**

**A Virtual Machine Platform for Non-Computer Professionals for Using Deep Learning to Classify Biological Sequences of Metagenomic Data**

**AUTHORS AND AFFILIATIONS:**

Zhencheng Fang<sup>1,2</sup>, Hongwei Zhou<sup>1,3\*</sup>

1 Microbiome Medicine Center, Department of Laboratory Medicine, Zhujiang Hospital, Southern Medical University, Guangzhou, PR China

2 Center for Quantitative Biology, Peking University, Beijing, PR China

3 State Key Laboratory of Organ Failure Research, Southern Medical University, Guangzhou, PR China

[fangzc@smu.edu.cn](mailto:fangzc@smu.edu.cn)

Correspondence:

Hongwei Zhou, [hzhou@smu.edu.cn](mailto:hzhou@smu.edu.cn)

**KEYWORDS:**

Metagenome, Microbiome, Sequence classification, Artificial intelligence, Deep learning, Algorithm design

**SUMMARY:**

This tutorial describes a simple method to construct a deep learning algorithm for performing 2-class sequence classification of metagenomic data.

**ABSTRACT:**

A variety of biological sequence classification tasks, such as species classification, gene function classification and viral host classification, are expected processes in many metagenomic data analyses. Since metagenomic data contain a large number of novel species and genes, high-performing classification algorithms are needed in many studies. Biologists often encounter challenges in finding suitable sequence classification and annotation tools for a specific task and are often not able to construct a corresponding algorithm on their own because of a lack of the necessary mathematical and computational knowledge. Deep learning techniques have recently become a popular topic and show strong advantages in many classification tasks. To date, many highly packaged deep learning packages, which make it possible for biologists to construct deep learning frameworks according to their own needs without in-depth knowledge of the algorithm details, have been developed. In this tutorial, we provide a guideline for constructing an easy-to-use deep learning framework for sequence classification without the need for sufficient mathematical knowledge or programming skills. All the code is optimized in a virtual machine so that users can directly run the code using their own data.

**INTRODUCTION:**

The metagenomic sequencing technique bypasses the strain isolation process and directly

sequences the total DNA in an environmental sample. Thus, metagenomic data contain DNA from different organisms, and most biological sequences are from novel organisms that are not present in the current database. According to different research purposes, biologists need to classify these sequences from different perspectives, such as taxonomic classification<sup>[1]</sup>, virus-bacteria classification<sup>[1-4]</sup>, chromosome-plasmid classification<sup>[1,5-7]</sup>, and gene function annotation (such as antibiotic resistance gene classification<sup>[8]</sup> and virulence factor classification<sup>[9]</sup>). Because metagenomic data contain a large number of novel species and genes, *ab initio* algorithms, which do not rely on known databases for sequence classification (including DNA classification and protein classification), are an important approach in metagenomic data analysis. However, the design of such algorithms requires professional mathematics knowledge and programming skills; therefore, many biologists and algorithm design beginners have difficulty constructing a classification algorithm to suit their own needs.

With the development of artificial intelligence, deep learning algorithms have been widely used in the field of bioinformatics to complete tasks such as sequence classification in metagenomic analysis. To help beginners understand deep learning algorithms, we describe the algorithm in an easy-to-understand fashion below.

An overview of a deep learning technique is shown in **Figure 1**. The core technology of a deep learning algorithm is an artificial neural network, which is inspired by the structure of the human brain. From a mathematical point of view, an artificial neural network may be regarded as a complex function. Each object (such as a DNA sequence, a photo or a video) is first digitized. The digitized object is then imported to the function. The task of the artificial neural network is to give a correct response according to the input data. For example, if an artificial neural network is constructed to perform a 2-class classification task, the network should output a probability score that is between 0-1 for each object. The neural network should give the positive object a higher score (such as a score higher than 0.5) while giving the negative object a lower score. To obtain this goal, an artificial neural network is constructed with the training and testing processes. During these processes, data from the known database are downloaded and then divided into a training set and test set. Each object is digitized in a proper way and given a label ("1" for positive objects and "0" for negative objects). In the training process, the digitized data in the training set are inputted into the neural network. The artificial neural network constructs a loss function that represents the dissimilarity between the output score of the input object and the corresponding label of the object. For example, if the label of the input object is "1" while the output score is "0.1", the loss function will be high; and if the label of the input object is "0" while the output score is "0.1", the loss function will be low. The artificial neural network employs a specific iterative algorithm that adjusts the parameters of the neural network to minimize the loss function. The training process finishes when the loss function cannot be obviously further decreased. Finally, the data in the test set are used to test the fixed neural network, and the ability of the neural network to calculate the correct labels for the novel objects is evaluated. More principles of deep learning algorithms can be found in the review in LeCun *et al.*<sup>[10]</sup>.

Although the mathematical principles of deep learning algorithms may be complex, many highly

packaged deep learning packages have recently been developed, and programmers can directly construct a simple artificial neural network with a few lines of code.

To assist biologists and algorithm design beginners in getting started in using deep learning more quickly, this tutorial provides a guideline for constructing an easy-to-use deep learning framework for sequence classification. This framework uses the “one-hot” encoding form as the mathematical model to digitize the biological sequences and uses a convolution neural network to perform the classification task (see the **Supplementary Material**). The only thing that the users need to do before using this guideline is to prepare four sequence files in “fasta” format. The first file contains all sequences of the positive class for the training process (referred to “p\_train.fasta”); the second file contains all sequences of the negative class for the training process (referred to “n\_train.fasta”); the third file contains all sequences of the positive class for the testing process (referred to “p\_test.fasta”); and the last file contains all sequences of the negative class for the testing process (referred to “n\_test.fasta”). The overview of the flowchart of this tutorial is provided in **Figure 2**, and more details will be mentioned below.

## PROTOCOL:

### 1. The installation of the virtual machine

1.1. Download the virtual machine file from (<https://github.com/zhenchengfang/DL-VM>).

1.2. Download the VirtualBox software from <https://www.virtualbox.org>.

1.3. Decompress the “.7z” file using related software, such as “7-Zip”, “WinRAR” or “WinZip”.

1.4. Install the VirtualBox software by clicking the **Next** button in each step.

1.5. Open the VirtualBox software and click the **New** button to create a virtual machine.

1.6. Step 6: Enter the specified virtual machine name in the “Name” frame, select **Linux** as the operating system in the “Type” frame, select **Ubuntu** in the “Version” frame and click the **Next** button.

1.7. Allocate the memory size of the virtual machine. We recommend that users pull the button to the right-most part of the green bar to assign as much memory as possible to the virtual machine, and then click the **Next** button.

1.8. Choose the **Use an existing virtual hard disk file** selection, select the file “VM\_Bioinfo.vdi” downloaded from Step 1.1 and then click the **Create** button.

1.9. Click the **Star** button to open the virtual machine.

NOTE: **Figure 3** shows the screenshot of the desktop of the virtual machine.

## **2. Create shared folders for files exchanging between the physical host and the virtual machine**

2.1. In the physical host, create a shared folder named “shared\_host”, and on the desktop of the virtual machine, create a shared folder named “shared\_VM”.

2.2. In the Menu Bar of the virtual machine, click **Devices, Shared Folder, Shared Folders Settings** successively.

2.3. Click the button in the upper right corner.

2.4. Select the shared folder in the physical host created in Step 2.1 and select the **Auto-mount** option. Click the **OK** button.

2.5. Restart the virtual machine.

2.6. Click the right click on the desktop of the virtual machine and open the terminal.

2.7. Copy the follow command to the terminal:

```
sudo mount -t vboxsf shared_host ./Desktop/shared_VM
```

2.7.1. When prompted for a password, enter “1” and hit the “**Enter**” key, as shown in **Figure 4**.

## **3. Prepare the files for the training set and test set**

3.1. Copy all four sequence files in “fasta” format for the training and testing process to the “shared\_host” folder of the physical host. In this way, all the files will also occur in the “shared\_VM” folder of the virtual machine. Then, copy the files in the “shared\_VM” folder to the “DeepLearning” folder of the virtual machine.

## **4. Digitize the biological sequences using “one-hot” encoding form**

4.1. Go to the “DeepLearning” folder, click the right click and open the terminal. Type the following command:

```
./onehot_encoding p_train.fasta n_train.fasta p_test.fasta n_test.fasta aa  
(for amino acid sequences)
```

*or*

```
./onehot_encoding p_train.fasta n_train.fasta p_test.fasta n_test.fasta nt  
(for nucleic acid sequences)
```

NOTE: A screenshot of this process is provided in **Figure 5**.

## 5. Train and test the artificial neural network

5.1. In the terminal, type the following command as shown in **Figure 6**:

```
python train.py
```

NOTE: The training process will begin.

### REPRESENTATIVE RESULTS:

In our previous work, we developed a series of sequence classification tools for metagenomic data using an approach similar to this tutorial<sup>[1,11,12]</sup>. As an example, we deposited the sequence files of the subset of training set and test set from our previous work<sup>[1,11]</sup> in the virtual machine.

Fang & Zhou<sup>11</sup> aimed to identify the complete and partial prokaryote virus virion proteins from virome data. The file “p\_train.fasta” contains the virus virion protein fragments for the training set; the file “n\_train.fasta” contains the virus nonvirion protein fragments for the training set; the file “p\_test.fasta” contains the virus virion protein fragments for the test set; and the file “n\_test.fasta” contains the virus nonvirion protein fragments for the test set. The user can directly execute the following two commands to construct the neural network:

```
./onehot_encoding p_train.fasta n_train.fasta p_test.fasta n_test.fasta aa
```

*and*

```
python train.py
```

The performance is shown in **Figure 7**.

Fang et al.<sup>3</sup> aimed to identify phage DNA fragments from bacterial chromosome DNA fragments in metagenomic data. The file “phage\_train.fasta” contains the phage DNA fragments for the training set; the file “chromosome\_train.fasta” contains the chromosome DNA fragments for the training set; the file “phage\_test.fasta” contains the phage DNA fragments for the test set; and the file “chromosome\_test.fasta” contains the chromosome DNA fragments for the test set. The user can directly execute the following two commands to construct the neural network:

```
./onehot_encoding phage_train.fasta chromosome_train.fasta phage_test.fasta
```

```
chromosome_test.fasta nt
```

*and*

```
python train.py
```

The performance is shown in **Figure 8**.

It is worth noting that because the algorithm contains some processes that have randomness, the above results may be slightly different if users rerun the script.

### FIGURE AND TABLE LEGENDS:

**Figure 1. Overview of the deep learning technique.**

**Figure 2. The overview of the flowchart of this tutorial.**

**Figure 3. The screenshot of the desktop of the virtual machine.**

**Figure 4. The screenshot of the activation of the shared folders.**

**Figure 5. The screenshot of the process of sequence digitization.**

**Figure 6. Train and test the artificial neural network.**

**Figure 7. The performance of prokaryote virus virion protein fragments identification.** The evaluation criteria are  $Sn=TP/(TP+FN)$ ,  $Sp=TN/(TN+FP)$ ,  $Acc=(TP+TN)/(TP+TN+FN+FP)$  and  $AUC$ .

**Figure 8. The performance of phage DNA fragments identification.** The evaluation criteria are  $Sn=TP/(TP+FN)$ ,  $Sp=TN/(TN+FP)$ ,  $Acc=(TP+TN)/(TP+TN+FN+FP)$  and  $AUC$ .

## **DISCUSSION:**

This tutorial provides an overview for biologists and algorithm design beginners on how to construct an easy-to-use deep learning framework for biological sequence classification in metagenomic data. This tutorial aims to provide intuitive understanding of deep learning and address the challenge that beginners often have difficulty installing the deep learning package and writing the code for the algorithm. For some simple classification tasks, users can use the framework to perform the classification tasks.

Considering that many biologists are not familiar with the command line of the Linux operating system, we preinstalled all the dependent software in a virtual machine. In this way, the user can directly run the code in the virtual machine following the protocol mentioned below. Additionally, if users are familiar with the Linux operating system and Python programming, they can also run this protocol directly on the server or local PC. In this way, the user should preinstall the following dependent software:

Python 2.7.12 (<https://www.python.org/>)

Python packages:

numpy 1.13.1 (<http://www.numpy.org/>)

h5py 2.6.0 (<http://www.h5py.org/>)

TensorFlow 1.4.1 (<https://www.tensorflow.org/>)

Keras 2.0.8 (<https://keras.io/>)

MATLAB	Component	Runtime	(MCR)	R2018a
( <a href="https://www.mathworks.com/products/compiler/matlab-runtime.html">https://www.mathworks.com/products/compiler/matlab-runtime.html</a> )				

The manual of our previous work<sup>3</sup> has a brief description of the installation. Note that the version number of each package corresponds to the version that we used in the code. The advantage of running the code in the server or local PC without the virtual machine is that the code can speed up with a GPU in this way, which can save much time in the training process. In this way, the user should install the GPU version of TensorFlow (see the manual of previous

work<sup>3</sup>).

Some of the critical steps within the protocol are described as follows. In step 3.1, the file names of “p\_train.fasta”, “n\_train.fasta”, “p\_test.fasta” and “n\_test.fasta” should be replaced by the used file names. The order of these four files in this command cannot be changed. If the files contain amino acid sequences, the last parameter should be “aa”; if the files contain nucleic acid sequences, the last parameter should be “nt”. This command uses the “one-hot” encoding form to digitize the biological sequences. An introduction of the “one-hot” encoding form is provided in the Supplementary Material. In step 5.1, because the virtual machine cannot be sped up with the GPU, this process may take a few hours or several days, depending on the data size. The progress bars for each iteration epoch are shown in the terminal. We set the number of epochs to 50, and thus, a total of 50 progress bars will be displayed when the training process is finished. When the test process is finished, the accuracy for the test set will be displayed in the terminal. In the “DeepLearning” folder of the virtual machine, a file named “predict.csv” will be created. This file contains all the prediction scores for the test data. The order of these scores corresponds to the sequence order in “p\_test.fasta” and “n\_test.fasta” (the first half of these scores corresponds to “p\_test.fasta”, while the second half of these scores corresponds to “n\_test.fatsa”). If users want to make predictions for the sequences whose true classes are unknown, they can also deposit these unknown sequences either in the “p\_test.fasta” or “n\_test.fasta” file. In this way, the scores of these unknown sequences will also be displayed in the “predict.csv” file, but the “accuracy” display in the terminal does not make sense. This script employs a convolutional neural network to perform the classification. The structure of the neural network and the code for the neural network are shown in the **Supplementary Material**.

One of the characteristics of deep learning is that many parameter settings require some experience, which can be a major challenge for beginners. To avoid beginner apprehension caused by a large number of formulas, we do not focus on the mathematical principles of deep learning, and in the virtual machine, we do not provide a special parameter setting interface. Although this may be a good choice for beginners, inappropriate parameter selection may also lead to a decline in precision. To allow beginners to better experience how to modify the parameters, in the script “train.py”, we add some comments to the related code, and users can modify the related parameters, such as the number of convolution kernels, to see how these parameters affect the performance.

Additionally, many deep learning programs should be run under a GPU. However, configuring the GPU also requires some computer skill that may be difficult for non-computer professionals; therefore, we choose to optimize the code in a virtual machine.

When solving other sequence classification tasks based on this guideline, users need only replace the four sequence files with their own data. For example, if users need to distinguish plasmid-derived and chromosome-derived sequences in metagenomic data, they can directly download plasmid genomes (<https://ftp.ncbi.nlm.nih.gov/refseq/release/plasmid/>) and bacterial chromosome genomes (<https://ftp.ncbi.nlm.nih.gov/refseq/release/bacteria/>) from

the RefSeq database and separate the genomes into a training set and test set. It is worth noting that DNA sequences in metagenomic data are often fragmented rather than complete genomes. In such cases, users can use the MetaSim<sup>[13]</sup> tool to extract the DNA fragment from the complete genome. MetaSim is a user-friendly tool with a GUI interface, and users can finish most operations using the mouse without typing any command on the keyboard. To simplify the operation for beginners, our tutorial is designed for a two-class classification task. However, we need to perform multiclassification in many tasks. In such cases, beginners can try to separate the multiclassification task into several two-class classification tasks. For example, to identify the phage host, Zhang *et al.* constructed 9 two-class classifiers to identify whether a given phage sequence can infect a certain host.

The homepage of this tutorial is deposited on the GitHub site <https://github.com/zhenchengfang/DL-VM>. Any update of the tutorial will be described on the website. Users can also raise their questions about this tutorial on the website.

#### **ACKNOWLEDGMENTS:**

This investigation was financially supported by the National Natural Science Foundation of China (81925026, 82002201, 81800746).

#### **DISCLOSURES:**

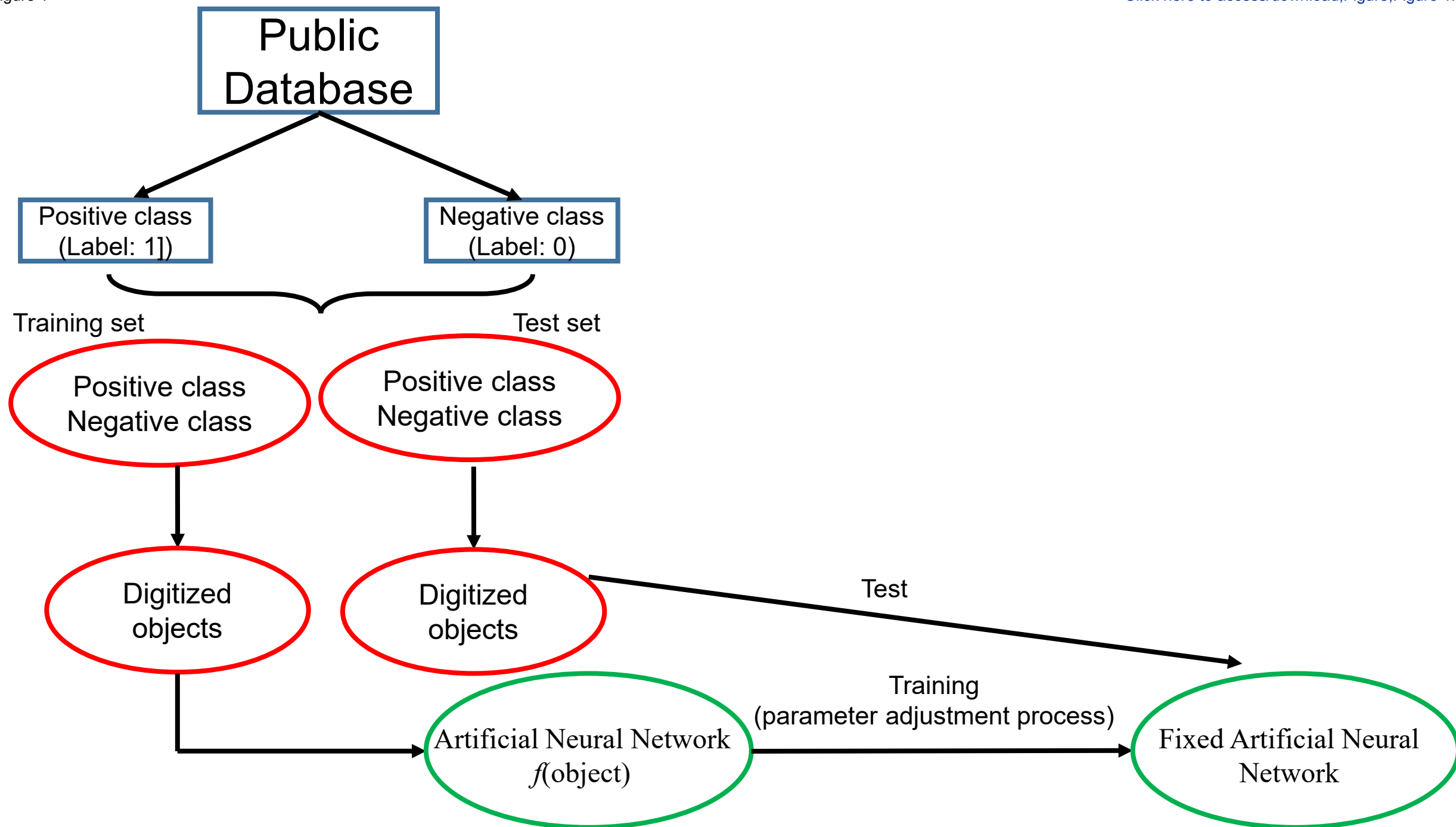
The authors declare that there are no conflicts of interest.

#### **REFERENCES:**

- [1] Liang, Q., Bible, P. W., Liu, Y., Zou, B., Wei, L. DeepMicrobes: taxonomic classification for metagenomics with deep learning. *NAR Genomics and Bioinformatics*. **2** (1) (2020).
- [2] Ren, J. et al. VirFinder: a novel k -mer based tool for identifying viral sequences from assembled metagenomic data. *Microbiome*. **5** (1), 69–69 (2017).
- [3] Fang, Z. et al. PPR-Meta: a tool for identifying phages and plasmids from metagenomic fragments using deep learning. *GigaScience*. **8** (6) (2019).
- [4] Ren, J. et al. Identifying viruses from metagenomic data using deep learning. *Quantitative Biology*. **8** (1), 64–77 (2020).
- [5] Zhou, F., Xu, Y. cBar: a computer program to distinguish plasmid-derived from chromosome-derived sequence fragments in metagenomics data. *Bioinformatics*. **26** (16), 2051–2052 (2010).
- [6] Krawczyk, P. S., Lipinski, L., Dziembowski, A. PlasFlow: predicting plasmid sequences in metagenomic data using genome signatures. *Nucleic Acids Research*. **46** (6) (2018).
- [7] Pellow, D., Mizrahi, I., Shamir, R. PlasClass improves plasmid sequence classification. *PLOS Computational Biology*. **16** (4) (2020).
- [8] Arango-Argoty, G. et al. DeepARG: a deep learning approach for predicting antibiotic resistance genes from metagenomic data. *Microbiome*. **6** (1), 1-15 (2018).
- [9] Zheng, D., Pang, G., Liu, B., Chen, L., Yang, J. Learning transferable deep convolutional neural networks for the classification of bacterial virulence factors. *Bioinformatics*. **36** (12), 3693-3702 (2020).
- [10] LeCun, Y., Bengio, Y., Hinton, G. Deep learning. *Nature*. **521** (7553), 436–444 (2015).

- 353 [11] Fang, Z., Zhou, H. VirionFinder: Identification of Complete and Partial Prokaryote Virus  
354 Virion Protein From Virome Data Using the Sequence and Biochemical Properties of Amino  
355 Acids. *Frontiers in Microbiology*. **12**, 615711 (2021).
- 356 [12] Fang, Z., Zhou, H. Identification of the conjugative and mobilizable plasmid fragments in  
357 the plasmidome using sequence signatures. *Microbial Genomics*. **6** (11) (2020).
- 358 [13] Richter, D.C., Ott, F., Auch, A.F., Schmid, R. Huson, D.H. MetaSim—a sequencing  
359 simulator for genomics and metagenomics. *PLoS One*. **3** (10), e3373 (2008).
- 360 [14] Zhang, M. et al. Prediction of virus-host infectious association by supervised learning  
361 methods. *BMC Bioinformatics*. **18** (3), 143-154 (2017).

Figure 1



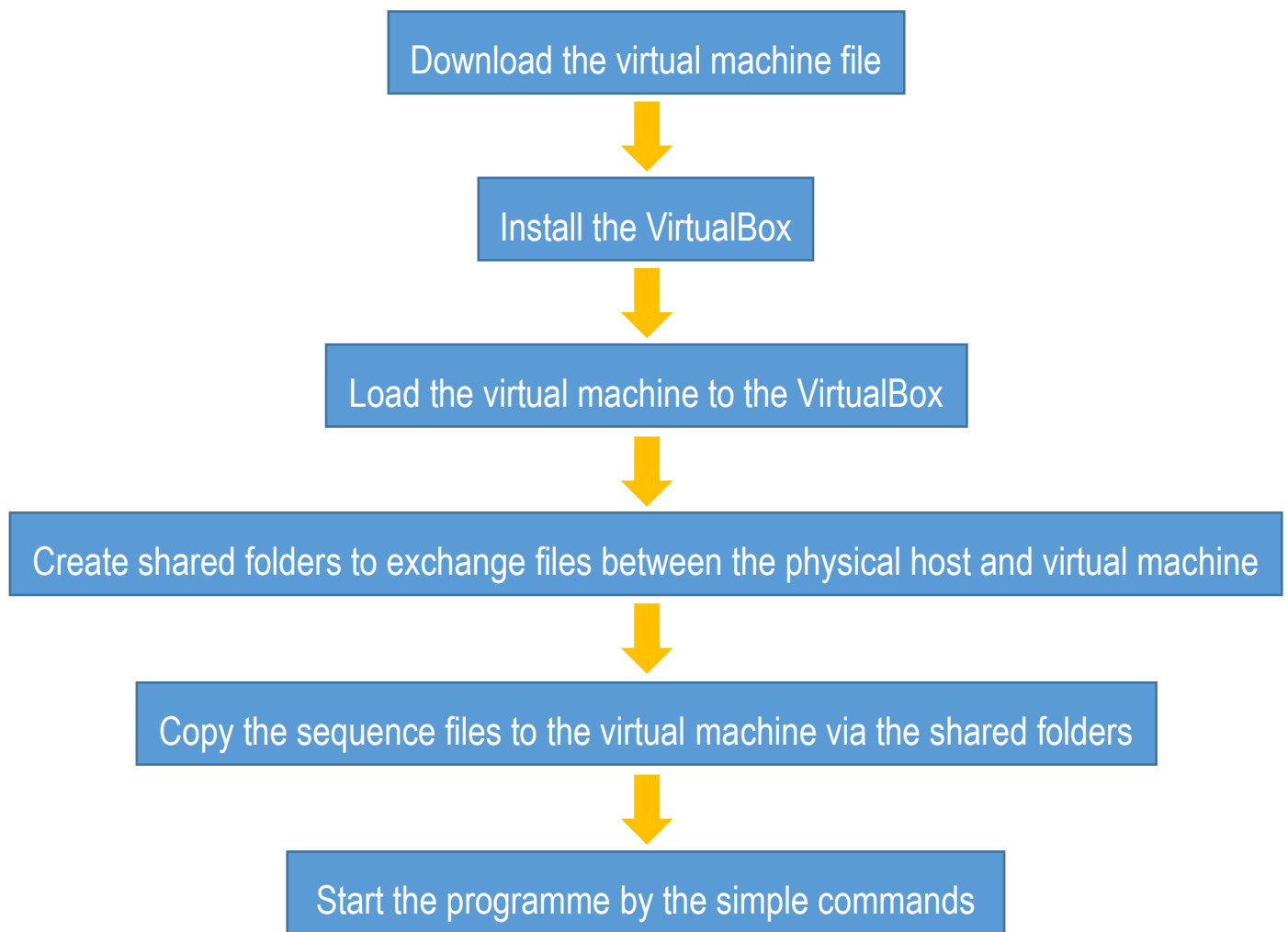


Figure 3

[Click here to access/download;Figure;Figure 3.png](#)

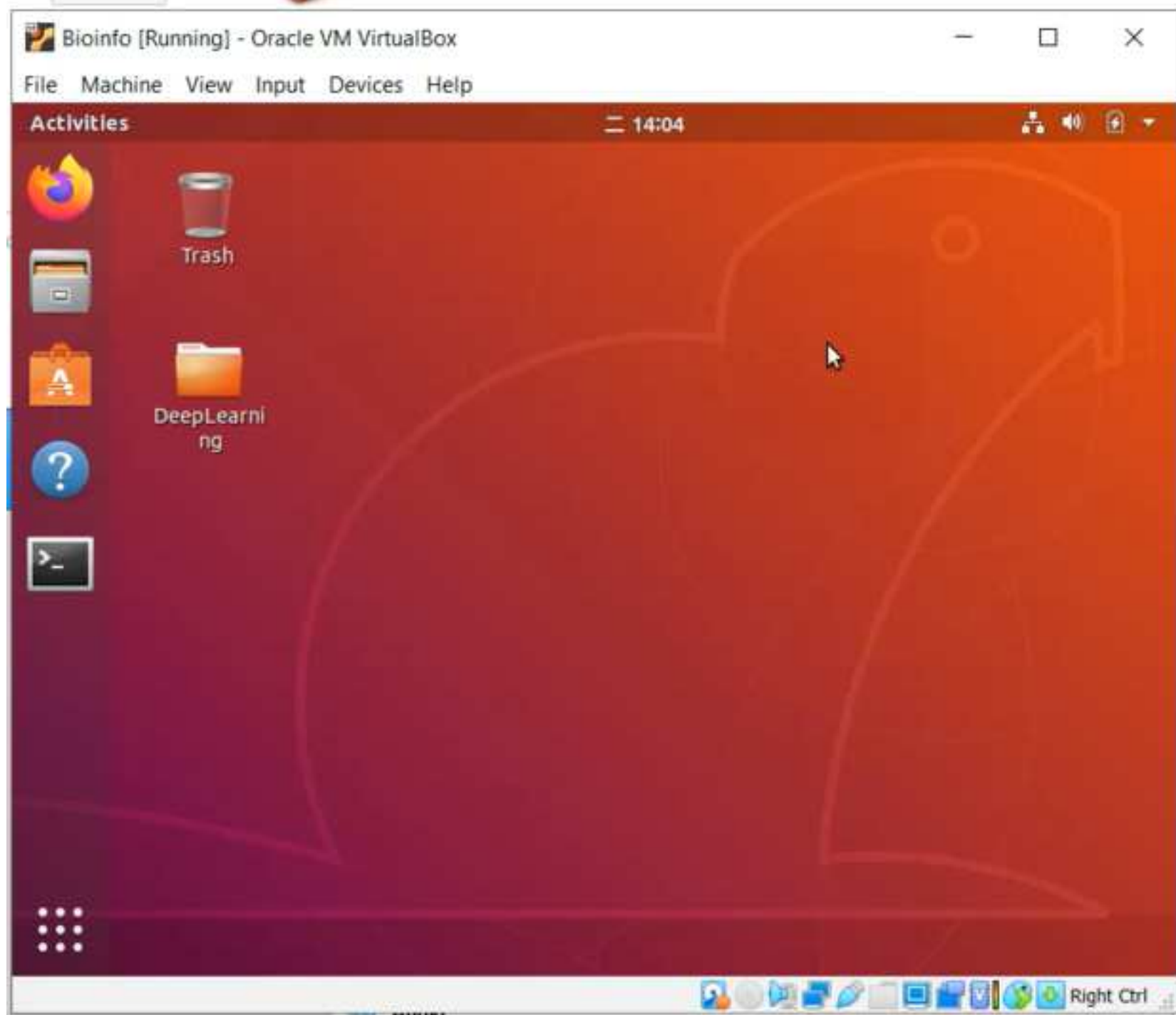


Figure 4

[Click here to access/download;Figure;Figure 4.png](#)

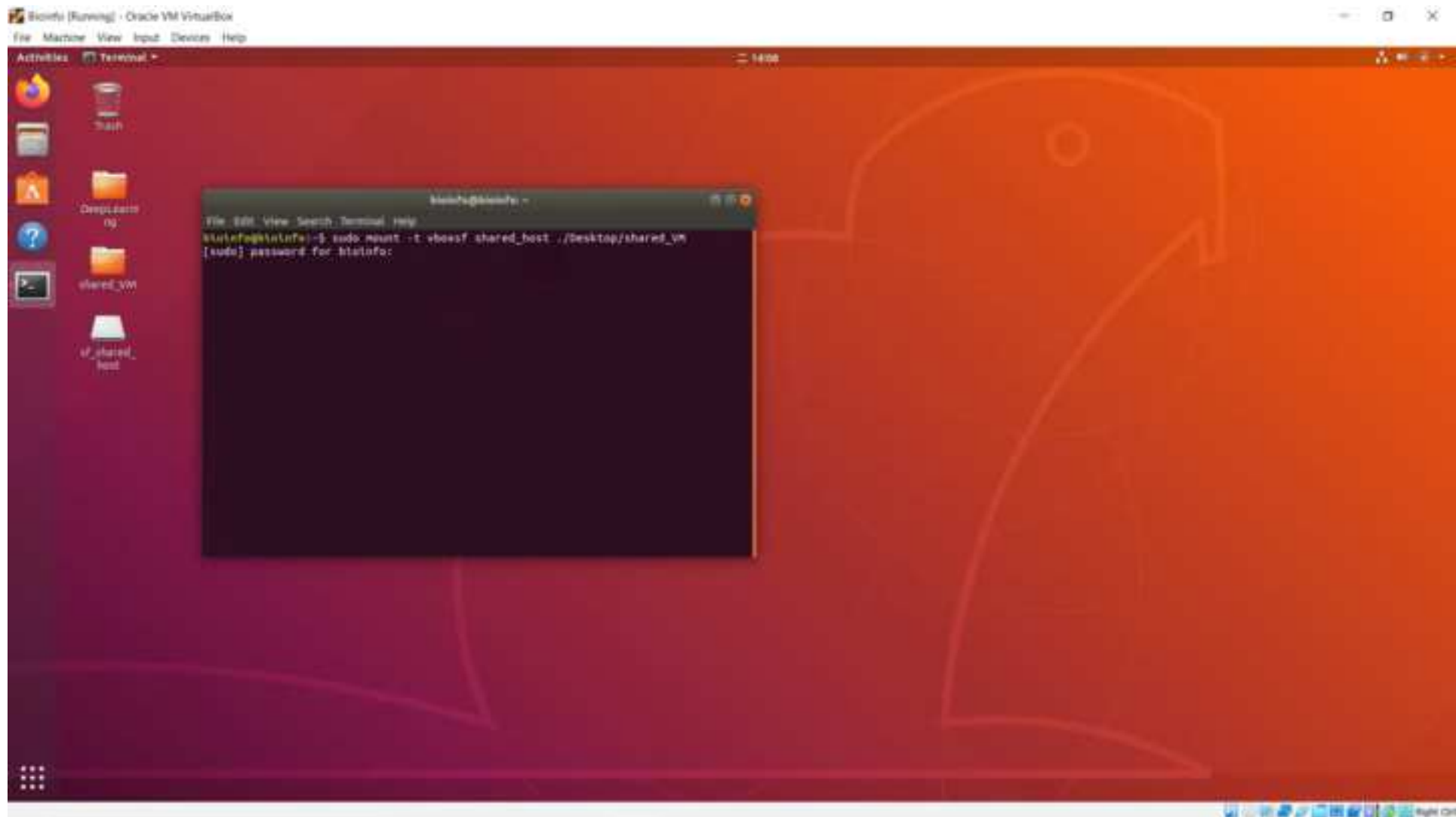


Figure 5

[Click here to access/download;Figure;Figure 5.png](#)

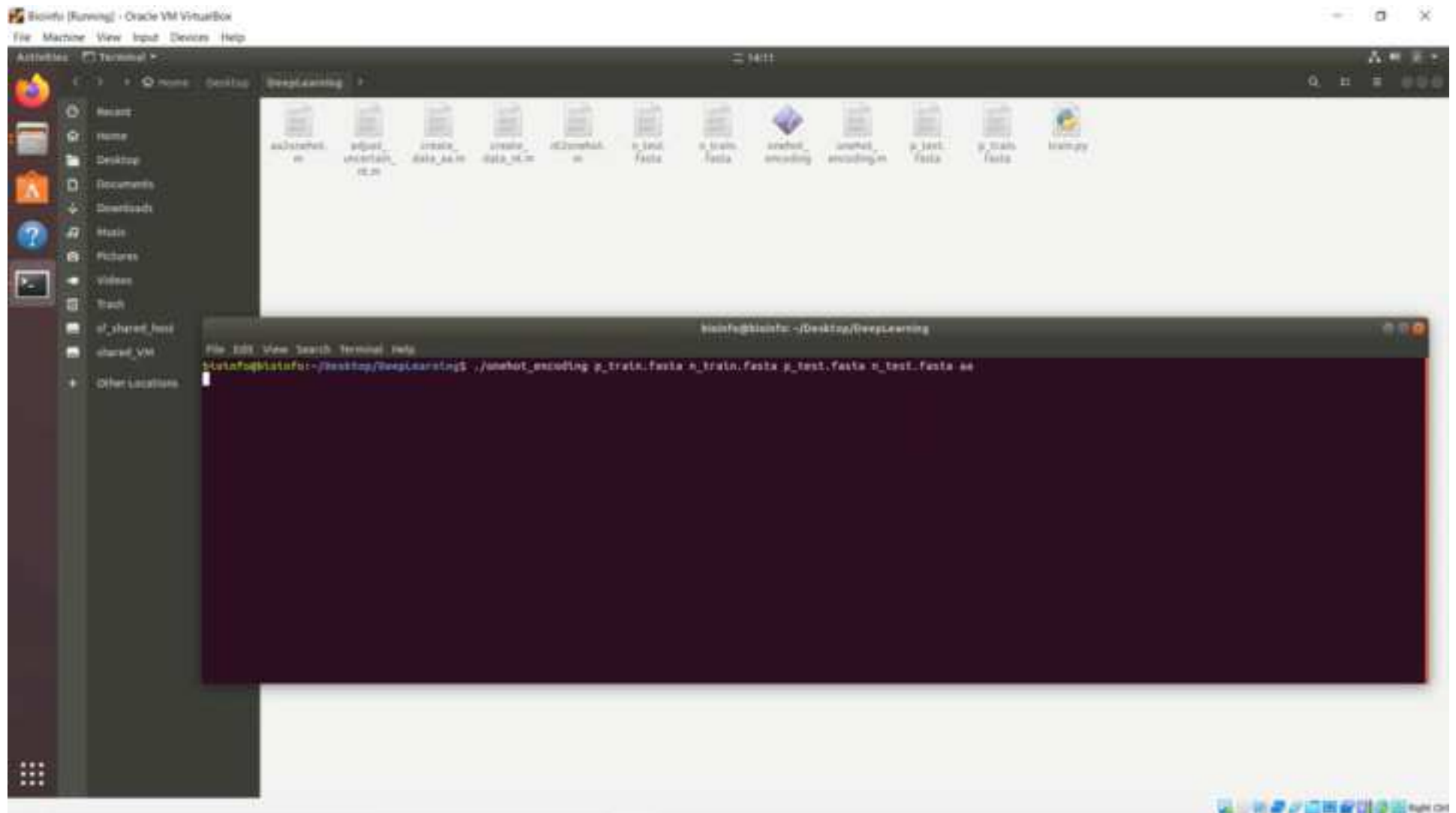


Figure 6

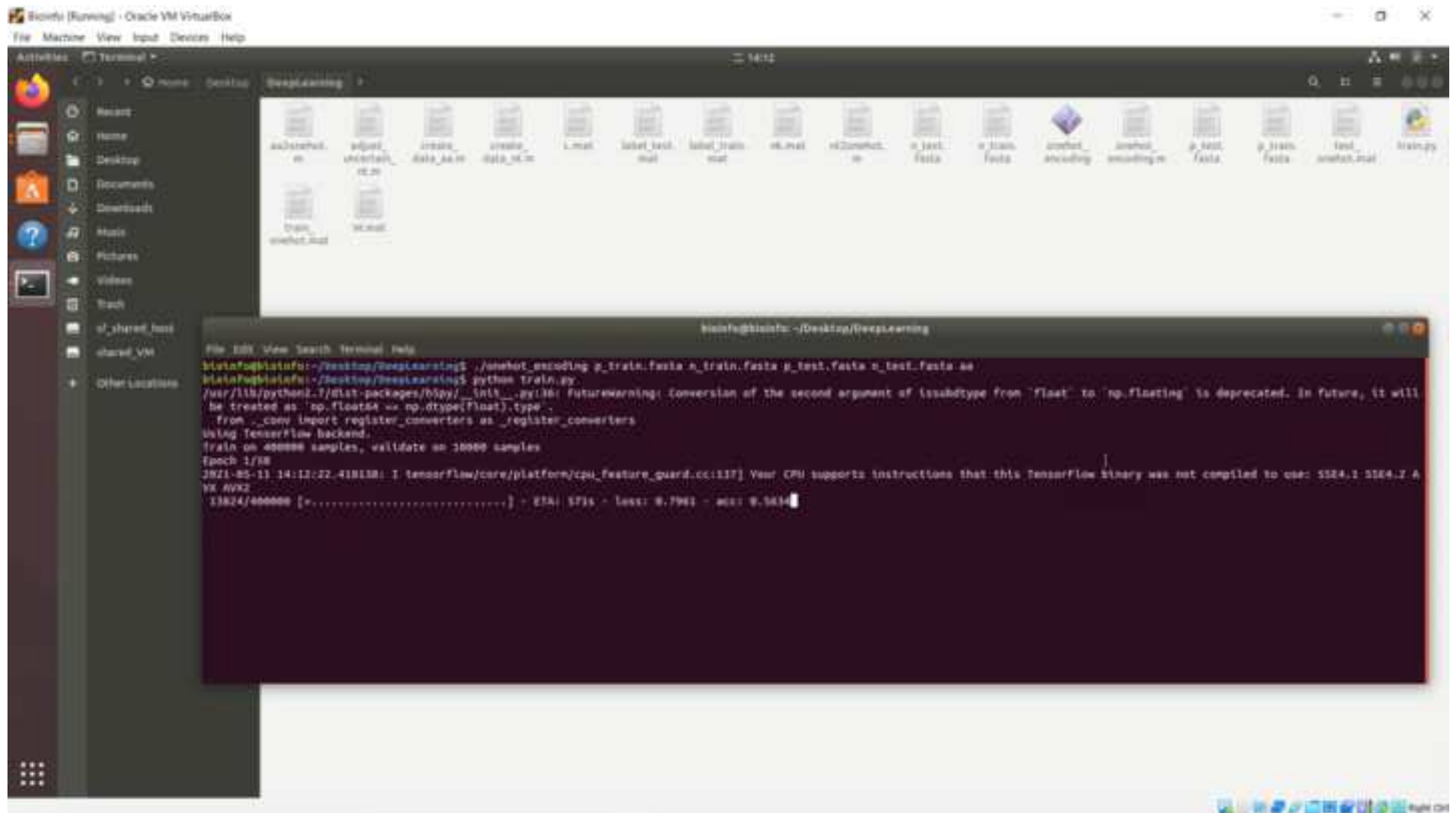
[Click here to access/download;Figure;Figure 6.png](#)

Figure 7

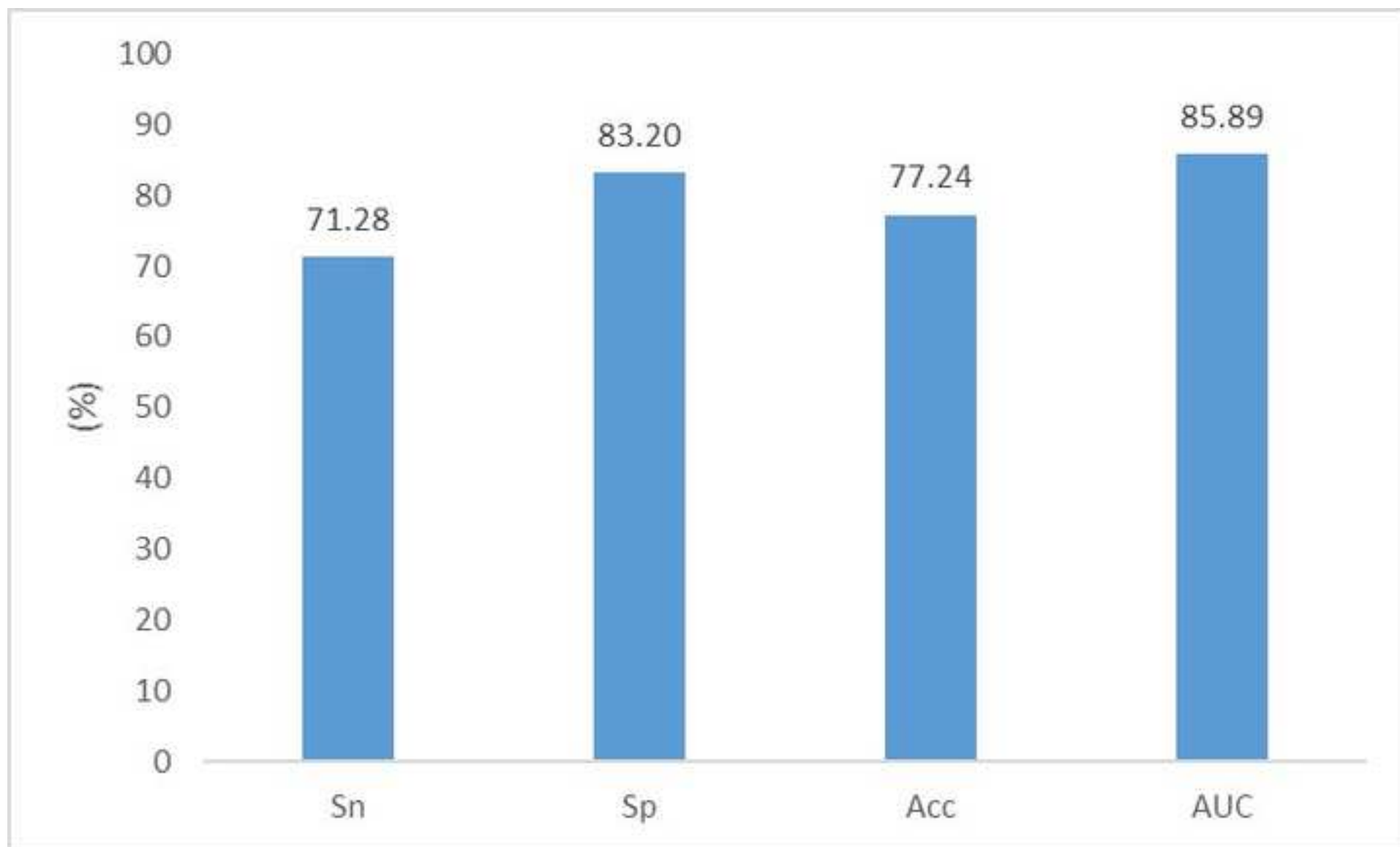
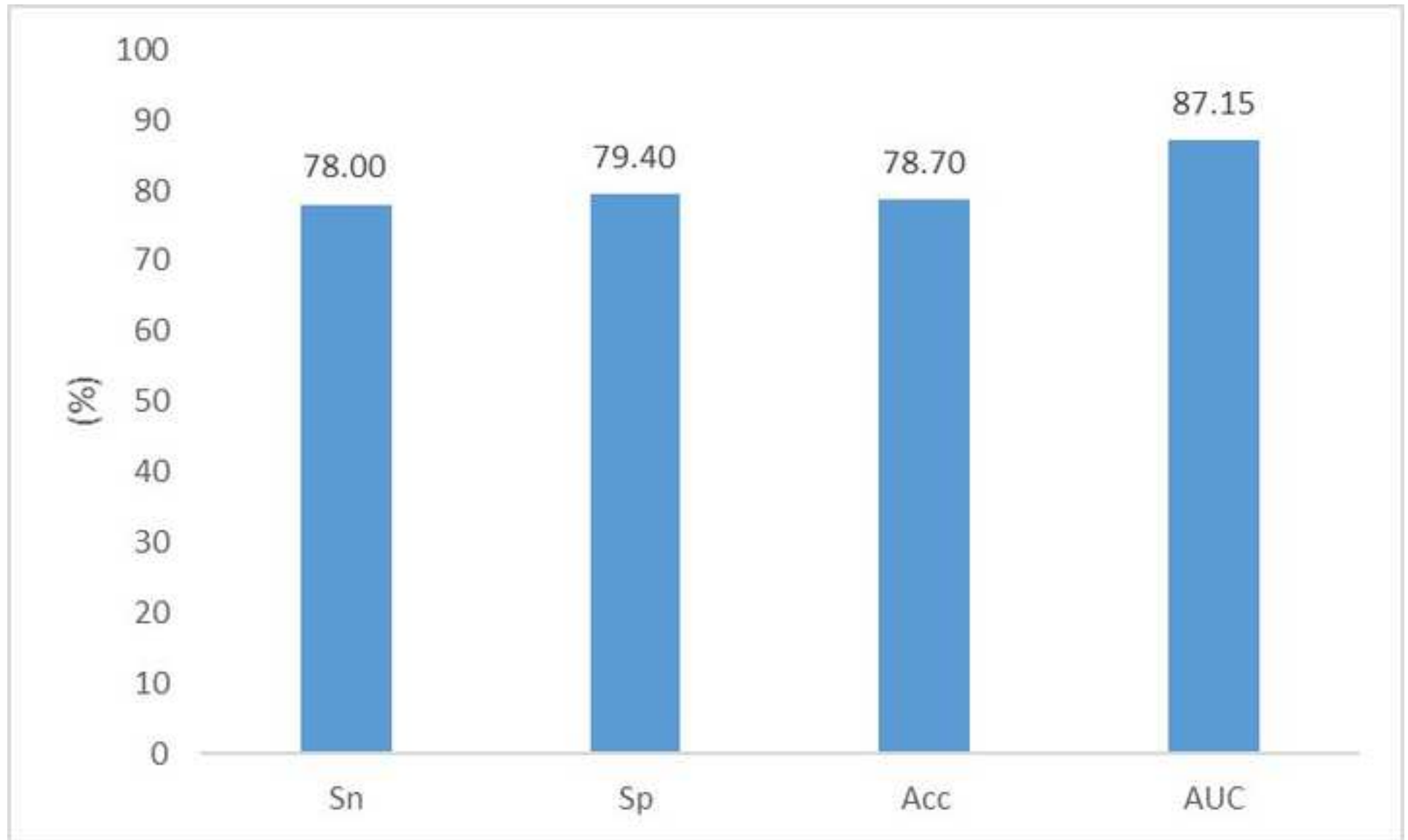


Figure 8





Click here to access/download

**Table of Materials**

JoVE\_Table\_of\_Materials.xlsx



## Cover Letter

Dear Editor,

Thank you very much for your previous email on July 24, 2021 regarding our manuscript “A virtual machine platform for non-computer professionals for using deep learning to classify biological sequences of metagenomic data” (Manuscript ID: JoVE62250R4). According to the editorial comments, we have modified our video as mentioned below.

### *1. Title Cards:*

- *Please add Standalone chapter title card for Introduction, Protocol, Result and Conclusion section.*
- *Main title card looks to be lower resolution. While fading in, it looks jarring. Please use high-quality title card and images in the Following timing:*

*00:01*

*00:09*

In the revised video, we have added the standalone chapter title card for Introduction, Protocol, Result and Conclusion section. And we have also used high-quality title card and images.

### *2. Video Editing Content:*

- *Please use cross dissolve or dip to black between two shots for smooth Transition.*

In the revised video, we have used cross dissolve between two shots of the PPT.

### *3. Audio Editing and Pacing:*

- *02:08 Please remove the background noise of room sound and mouse clicking sound from the Video*

The background noise and mouse clicking sound has also been removed.

Moreover, although we selected the option of making the video by ourselves in the submission system, we were willing to consider making the video by the journal team if our video hard to meet the journal requirement.

Since we are not able to open the link <https://www.dropbox.com/request/kOhRq6A5kiLIXR5ihTn1?oref=e>, we therefore uploaded the video to <https://www.jove.com/account/file-uploader?src=18986298>.

We hope that the above revisions and point-by-point responses have clarified all the points, and we hereby resubmit our manuscript to JoVE. We thank you for your kind consideration.

Sincerely,

Hongwei Zhou, Ph. D., Professor

Microbiome Medicine Center, Division of Laboratory Medicine, Zhujiang Hospital,  
Southern Medical University, Guangzhou, 510282, China

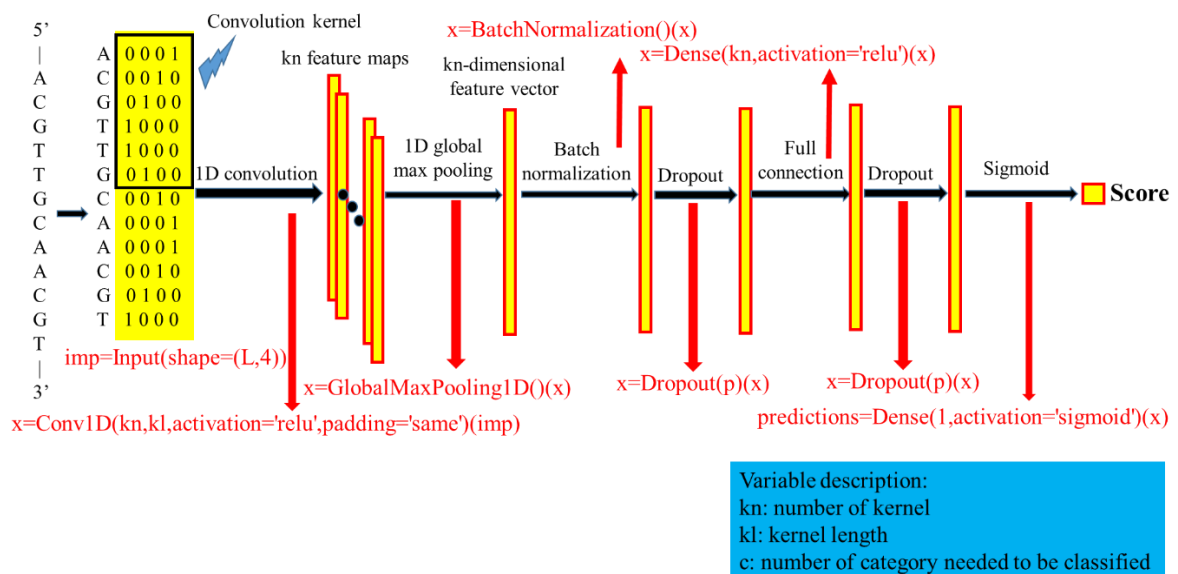
Email: [hzhou@smu.edu.cn](mailto:hzhou@smu.edu.cn)

## 1 The “one-hot” encoding form

We introduce a simple digitization method for biological sequences, namely, one-hot encoding. For a DNA sequence, bases A, C, G and T can be represented by one-hot vectors of [0,0,0,1], [0,0,1,0], [0,1,0,0] and [1,0,0,0], respectively. Such one-hot vectors are connected according to the base order in the DNA sequence. Therefore, a DNA sequence of length  $L$  can be represented by a base one-hot matrix with length  $L$  and width 4. If the complementary strand is included in the matrix, the matrix will have a length of  $2L$  and width of 4. Similarly, for the protein sequence, each amino acid can be represented by a one-hot vector containing 20 bits, in which 19 bits are 0 and a specific bit is 1. For example, amino acid A can be represented as [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1], amino acid C can be represented as [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0] and so on. In this way, a protein sequence with length  $L$  can be represented by an amino acid matrix of length  $L$  and width 20.

## 2 Artificial neural network construction

The construction methods for artificial neural networks are diverse. Herein, we introduce a simple construction method. Figure S1 shows the structure of an artificial neural network. This structure is easily implemented with a few lines of code (shown in red text in Figure S1) in the deep learning package Keras.



**Figure S1. Construction of the artificial neural network.** The red text represents the Keras code for the corresponding layer in the neural network. The neural network takes a biological sequence as input and outputs a score between 0-1. A higher score means that the input sequence is more likely to be a positive class.

With the structure shown in Figure S1, the programmer should first specify the size of the input one-hot matrix. For the example in Figure S1, the DNA sequence is represented by a base one-hot matrix; in such cases, the corresponding Keras code is:

```
imp=Input(shape=(L,4))
```

where  $L$  represents the length of the one-hot matrix (the sequence length). If the corresponding matrix is an amino acid one-hot matrix, the second parameter of 4 should be set to 20.

Then, a 1D convolution operation is performed over the one-hot matrix. In such operations, multiple convolution kernels scan the one-hot matrix, and each kernel will generate a feature map. The code for the 1D convolution layer in Keras is:

$$x=\text{Conv1D}(kn,kl,\text{activation}='relu',\text{padding}='same')(imp)$$

where  $kn$  represents the number of kernels and  $kl$  represents the length of the kernel. The convolution operation can be interpreted as eyes scanning over a picture. The convolution kernel is equivalent to an eye, the number of kernels is equivalent to the number of eyes, and the kernel length is equivalent to the vision field of each eye. It is worth noting that higher  $kn$  and  $kl$  values are not necessarily better. The “relu” parameter is a nonlinear function that performs a nonlinear transformation on the one-hot matrix. The nonlinear transformation can be viewed as similar to human thought processes.

After the 1D convolution, a 1D global max pooling operation is performed on each feature map. In this process, each feature map is represented by its maximum value. In this way, the  $kn$  feature maps can be reduced to a  $kn$ -dimensional feature vector. The code of the 1D global max pooling layer in Keras is:

$$x=\text{GlobalMaxPooling1D}()(x)$$

The  $kn$ -dimensional feature vector then goes through a batch normalization layer. The code of the batch normalization layer in Keras is:

$$x=\text{BatchNormalization}()(x)$$

The  $kn$ -dimensional feature vector then goes through a “dropout” layer, which can prevent overfitting in the training process. The code of the “dropout” layer in Keras is:

$$x=\text{Dropout}(p)(x)$$

where  $p$  represents the dropout probability.

The  $kn$ -dimensional feature vector then goes through a fully connected layer, which can be viewed as deep thinking in a human brain. The code of the fully connected layer in Keras is:

$$x=\text{Dense}(kn,\text{activation}='relu')(x)$$

Again, the  $kn$ -dimensional feature vector then goes through a “dropout” layer. The code of the “dropout” layer in Keras is:


$$x=\text{Dropout}(p)(x)$$

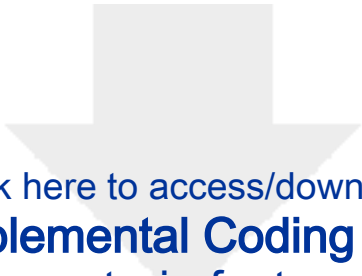
Finally, the feature vector goes through a sigmoid layer that calculates the score for the input DNA. The code for the sigmoid layer in Keras is:

$$\text{predictions}=\text{Dense}(1,\text{activation}='sigmoid')(x)$$

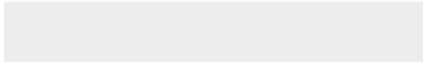


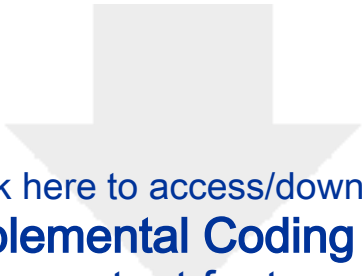
Click here to access/download  
**Supplemental Coding Files**  
p\_train.fasta



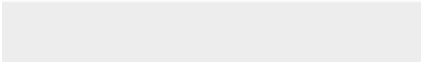



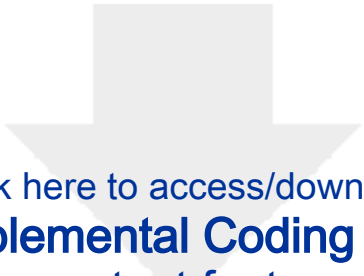
Click here to access/download  
**Supplemental Coding Files**  
n\_train.fasta



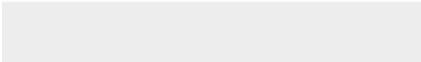



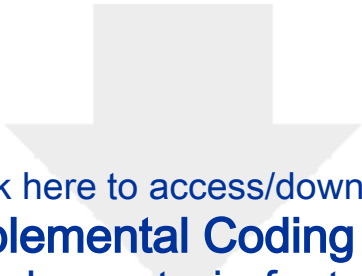
Click here to access/download  
**Supplemental Coding Files**  
p\_test.fasta



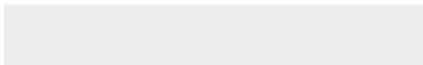




Click here to access/download  
**Supplemental Coding Files**  
n\_test.fasta



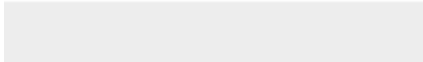



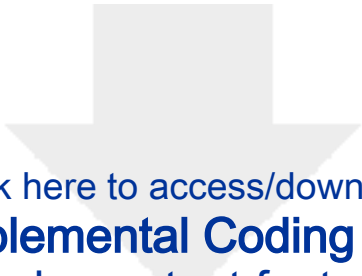
Click here to access/download  
**Supplemental Coding Files**  
phage\_train.fasta



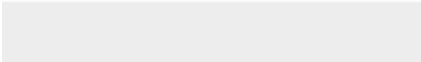



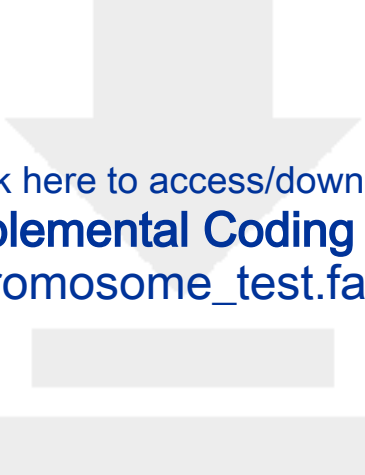
Click here to access/download  
**Supplemental Coding Files**  
chromosome\_train.fasta





Click here to access/download  
**Supplemental Coding Files**  
phage\_test.fasta





[Click here to access/download](#)  
**Supplemental Coding Files**  
chromosome\_test.fasta