Flow is composed of a bunch of packets with the same five-tuple:

1. the source address (sa)
2. the destination address(da)
3. the source port (for TCP and UDP traffic) (pr)
4. the destination port (sp)
5. and the protocol number (dp)

The the packet of a flow move in the same direction (unidirectional). The bidirectional flows consists of a pair of unidirectional flows whose source and destination address and ports are reversed, and whose time spans overlap. A bidirectional flow is defined as a section.

The following table list each element in allSamples.csv. (The dimension of X\_train is (31662, 885), i.e., there are totally 31662 samples, and each sample has 885 features.)

In the following table, from “sa” to “entropy”, there are totally ***703*** features. (Plus type, label, file\_name, and index, totally there are 707 columns.)

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **name** | **Data element** | **Notes** |
| 1 | file\_name | Json file name | scpUp2: |
| 2 | label |  |  |
| 3 | type | Traffic type, defined based on Json file names  ***E.g, DL output / label*** |  |
| 4 | sa | Source IP Address | Internet Protocol |
| 5 | da | Destination IP Address |  |
| 6 | sp | Source Port | TCP or UDP |
| 7 | dp | Destination Port |  |
| 8 | pr | Protocol Number |  |
| 9 | tls\_scs | Transport layer security |  |
| 10 | tls\_ext\_server\_name |  |  |
| 11 | tls\_c\_key\_length |  |  |
| 12 | http\_content\_type |  |  |
| 13 | http\_user\_agent |  |  |
| 14 | http\_accept\_language |  |  |
| 15 | http\_server |  |  |
| 16 | http\_code |  |  |
| 17 | dns\_domain\_name |  |  |
| 18 | dns\_ttl |  |  |
| 19 | dns\_num\_ip |  |  |
| 20 | dns\_domain\_rank |  |  |
| 21 | formean |  |  |
| 22 | forvar |  |  |
| 23 | backmean |  |  |
| 24 | backvar |  |  |
| 25 | duration |  |  |
| 26 | tot\_forpkts |  |  |
| 27 | tot\_backpkts |  |  |
| 28 | tot\_forpktsize |  |  |
| 29 | tot\_backpktsize |  |  |
| 30 | maxforpktsize |  |  |
| 31 | minforpktsize |  |  |
| 32 | maxbackpktsize |  |  |
| 33 | minbackpktsize |  |  |
| 34 | numbytepersec |  |  |
| 35 | foriptmean |  |  |
| 36 | foriptstd |  |  |
| 37 | backiptmean |  |  |
| 38 | backiptstd |  |  |
| 39 | totfoript |  |  |
| 40 | totbackipt |  |  |
| 41 | maxfoript |  |  |
| 42 | minfoript |  |  |
| 43 | maxbackipt |  |  |
| 44 | minbackipt |  |  |
| 45 | numforpktpersec |  |  |
| 46 | numbackpktpersec |  |  |
| 47 | numpktpersec |  |  |
| 48 | ttlout |  |  |
| 49 | ttlin |  |  |
| 50 ~ 449 | splt20\*20 |  |  |
| 450 ~ 705 | dist0~255 |  |  |
| 706 | entropy |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  | bytes\_out | Number of bytes sa→da  in TCP, UDP, ICMP, or IP Data fields | Any Protocol |
|  | num\_pkts\_out | Number of packets sa→da |  |
|  | time\_start | Start time, seconds since epoch |  |
|  | time\_end | End time, seconds since epoch |  |
|  | packets | Array of packet lengths, directions, and times |  |
|  | bytes\_in | Number of bytes sa←da  in TCP, UDP, ICMP, or IP Data fields | Bidirectional flows only |
|  | num\_pkts\_in | Number of packets sa←da |  |
|  | expire\_type | Expiration type | "i" denotes an inactive expiration  "a" denotes an active expiration. |
|  | byte\_dist | a count of the number of occurrences of each byte value in the Data fields of a flow | A 16\*16 matrix (256 integers), with the nth element of the array corresponding to the byte value n. |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# How to run the project (6/2/2022)

In Linux

Data Preparation

1. pcap file: Capture flow using Wireshark and tcpdump. Save them as .pcap data.
2. json file: Convert pacp file to json using joy.

Install joy:

1) clone joy 2.0 from <https://github.com/cisco/joy> to Home folder, i.e., the directory is *home/ir/joy.*

git clone https://github.com/cisco/joy.git

2) In terminal, first install dependencies:

sudo apt-get install build-essential libssl-dev libpcap-dev libcurl4-openssl-dev

3) Configure

cd joy

[joy]$ ./configure --enable-gzip

4) build

[joy]$ make clean;make

Reference: <https://github.com/cisco/joy/wiki/Building>

Run pcap\_to\_json.sh to convert a PCAP file to a json file

cd data

sh pcap\_to\_json.sh PCAP tmpJSON JSON 50

1. csv file: Calculate the features associated with each flow using NetMate or ISCXFlowMeter. Generate .csv data.

cd prepro

sh multi\_gen.sh ../data/JSON Table.csv 0

python3 toTrain.py Table.csv

1. To run train.py, in Vscode terminal, choose “python 3.8.8 (‘base’: conda) interpreter, then in terminal, type:

conda activate tf-gpu

cd ./main

python3 train.py

or

python3 train.py --mode DNN --source\_data\_folder ../data --output\_folder ./output --batch\_size 1024 --patience 1000

data download from the author:

X\_train: (31662, 885)

# Modeling

In prepro folder, generate Table\_v1. Instead of using packet statistic features in Table\_v0, Table\_v1 directly uses packet information. In prepro folder, execute:

sh multi\_gen.sh ../data/JSON Table\_v1.csv 0

Basic\_Info(data, i, df, Type, title)

TLS(data, i, df)

#http(data, i ,df)

dns(data, i ,df)

ttl(data, i, df)

#statisticfeature(data,i ,df)

#iptfeature(data, i, df)

Byte\_dist(data, i, df)

Marcov(data, i, df)

packets(data, i, df)

if(opts.is\_malware == 1):

df.loc[i,'Malware\_label'] = 1

else:

df.loc[i,'Malware\_label'] = 0

Using Table\_v1, different features are selected as neural network input.

## MLP Scheme 2:

In toTrain.py (line 128), change code to:

X\_train = table\_s2.values

In prepro folder, execute:

python3 toTrain.py Table\_v1.csv

In main folder, set neural network structure as:

MLP: 1600 (0.5) – 1200 (0.5) – 880 (0.25) - 640 (0.25) - 360 (0.25) - 220 (0.2) - 80 (0.2)

execute:

sh TRAIN.sh DNN

Result:

X\_train dimension: (32861, 1003)

## MLP Scheme 3:

In toTrain.py (line 128), change code to:

X\_train = table\_s3.values

In prepro folder, execute:

python3 toTrain.py Table\_v1.csv

In main folder, set neural network structure as

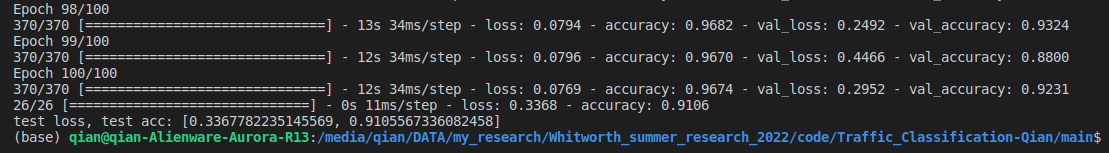
MLP: 1600 (0.5) – 1200 (0.5) – 880 (0.25) - 640 (0.25) - 360 (0.25) - 220 (0.2) - 80 (0.2)

execute:

sh TRAIN.sh DNN

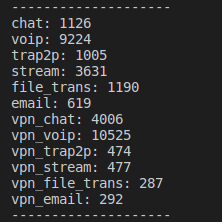
Result:

X\_train dimension: (32861, 1341)



# August 18th, 2022

Dateset:



Modified Table\_Generator function and generated a table of 420 columns:

'file\_name','Malware\_label','type','sa','da','sp','dp','pr','tls\_scs','tls\_ext\_server\_name','tls\_c\_key\_length', 'ttlout', 'ttlin','dist\_0', …, 'dist\_255', 'entropy', 'packet\_0\_b', 'packet\_0\_ipt', 'packet\_0\_dir', …, 'packet\_49\_b', 'packet\_49\_ipt', 'packet\_49\_dir'

In prepro folder, generate Table\_v2. Instead of using packet statistic features in Table\_v0, Table\_v2 directly uses packet information. Moreover, dns and Marcov distance features are removed.

In prepro folder, execute:

sh multi\_gen.sh ../data/JSON Table\_v2.csv 0

df = pd.DataFrame(columns = col)

Basic\_Info(data, i, df, Type, title)

TLS(data, i, df)

#http(data, i ,df)

#dns(data, i ,df)

ttl(data, i, df)

#statisticfeature(data,i ,df)

#iptfeature(data, i, df)

Byte\_dist(data, i, df)

#Marcov(data, i, df)

packets(data, i, df)

if(opts.is\_malware == 1):

df.loc[i,'Malware\_label'] = 1

else:

df.loc[i,'Malware\_label'] = 0

Using Table\_v2, different features are selected as neural network input.

## MLP Scheme 1:

In toTrain.py (line 128), change code to:

X\_train = table\_s1.values

execute (in prepro folder):

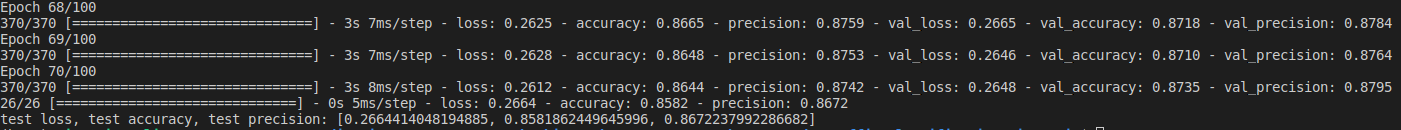
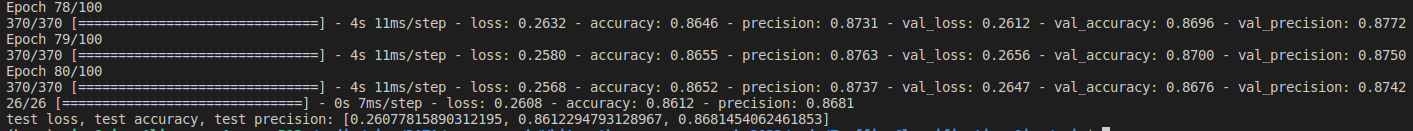
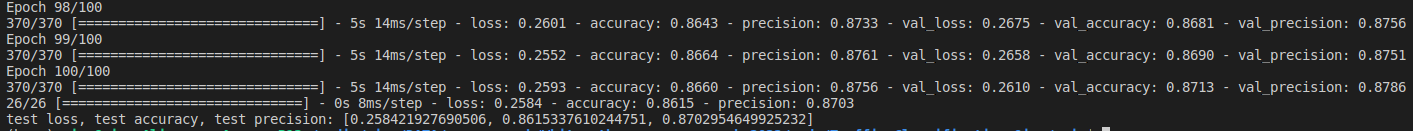
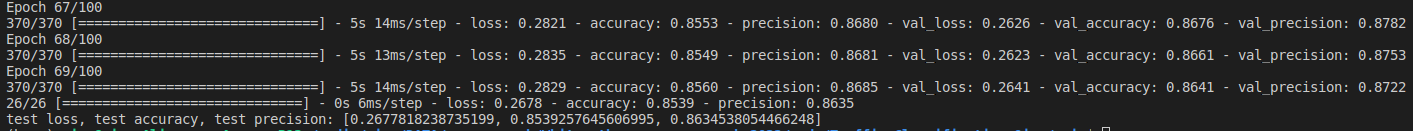
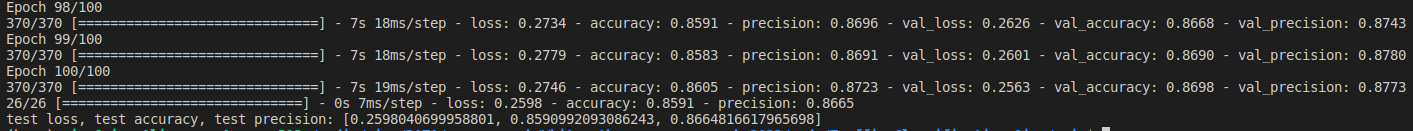
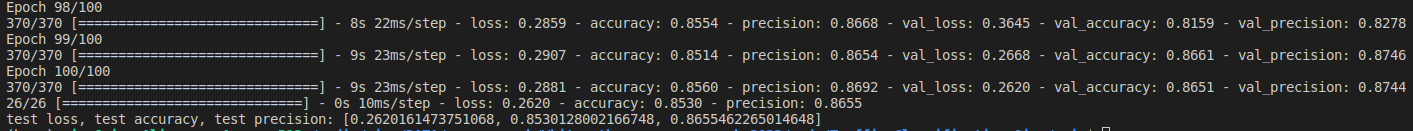
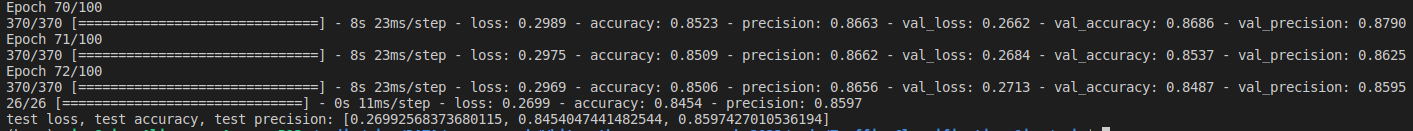
python3 toTrain.py Table\_v2.csv

To train the neural network, execute (in main folder):

sh TRAIN.sh DNN

Result:



1. 5/16/2023 More data for the paper
2. In main folder -->train.py, set neural network structure as:
3. DNN 1.3 (3 hidden layers)
4. MLP: 1600 (0.5) – 1200 (0.25) – 220 (0.2)
5. 
6. DNN 1.4 (4 hidden layers)
7. MLP: 1600 (0.5) – 1200 (0.25) – 640 (0.25) – 120 (0.2)
8. 
9. DNN 1.5 (5 hidden layers) (best)
10. MLP: 1600 (0.5) – 1200 (0.5) – 880 (0.25) – 640 (0.25) – 220 (0.2)
11. 
12. DNN 1.6 (6 hidden layers)
13. MLP: 1600 (0.5) – 1200 (0.5) – 880 (0.25) – 360 (0.25) – 220 (0.2) – 80 (0.2)
14. 
15. DNN 1.7 (7 hidden layers)
16. MLP: 1600 (0.5) – 1200 (0.5) – 880 (0.25) – 640 (0.25) – 360 (0.25) – 220 (0.2) – 80 (0.2)
17. 
18. DNN 1.9 (9 hidden layers)
19. MLP: 1600 (0.5) – 1200 (0.5) – 1200 (0.25) – 880 (0.25) – 640 (0.25) – 640 (0.25) – 360 (0.25) – 220 (0.2) – 80 (0.2)
20. 
21. DNN 1.10 (10 hidden layers)
22. 1600 (0.5) – 1600 (0.5) – 1200 (0.5) – 1200 (0.25) – 880 (0.25) – 640 (0.25) – 640 (0.25) – 360 (0.25) – 220 (0.2) – 80 (0.2)
23. 

## MLP Scheme 2:

In toTrain.py (line 128), change code to:

X\_train = table\_s2.values

execute (in prepro folder):

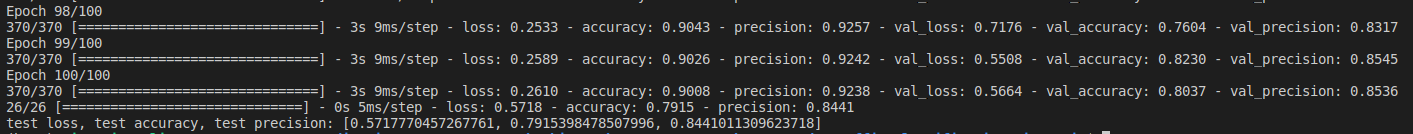
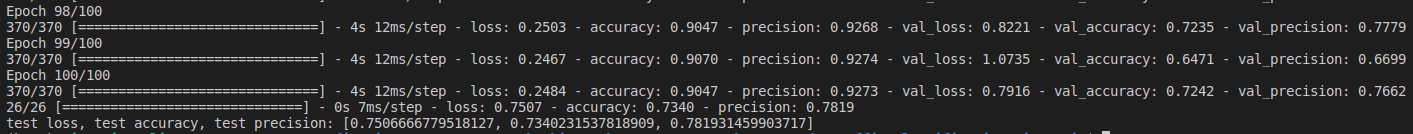
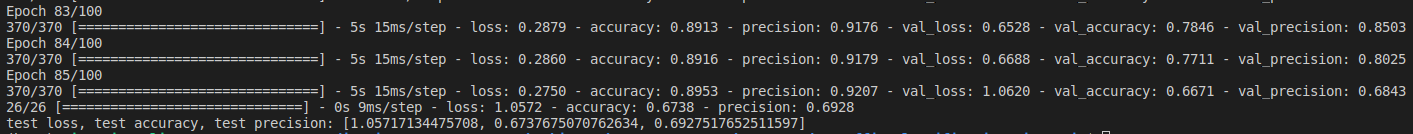
python3 toTrain.py Table\_v2.csv

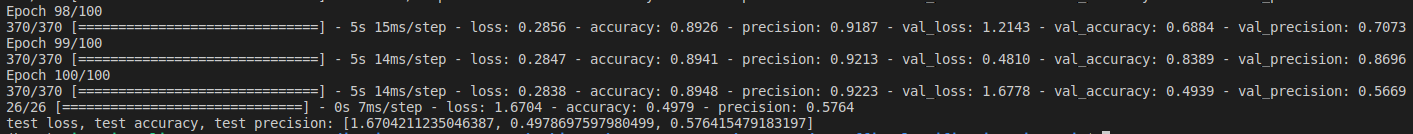
To train the neural network, execute (in main folder):

sh TRAIN.sh DNN

Result:

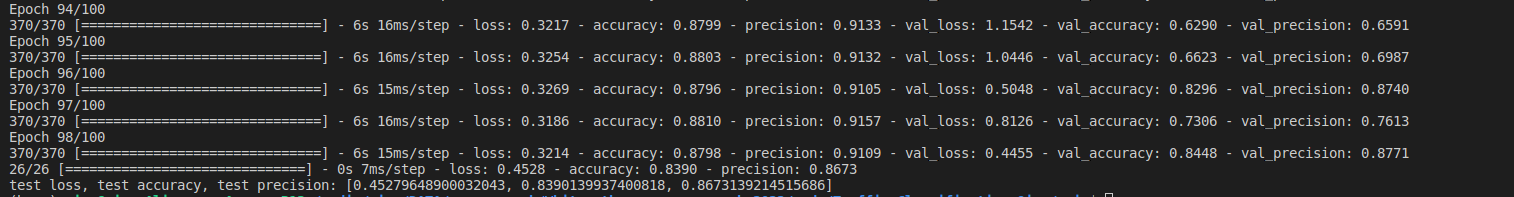


1. 5/16/2023 More data for the paper
2. In main folder -->train.py, set neural network structure as:
3. DNN 2.3 (3 hidden layers)
4. MLP: 1600 (0.5) – 1200 (0.25) – 220 (0.2)
5. 
6. DNN 2.4 (4 hidden layers)
7. MLP: 1600 (0.5) – 1200 (0.25) – 640 (0.25) – 120 (0.2)
8. 
9. DNN 2.5 (5 hidden layers) (best)
10. MLP: 1600 (0.5) – 1200 (0.5) – 880 (0.25) – 640 (0.25) – 220 (0.2)
11. 
12. DNN 2.6 (6 hidden layers)
13. MLP: 1600 (0.5) – 1200 (0.5) – 880 (0.25) – 360 (0.25) – 220 (0.2) – 80 (0.2)

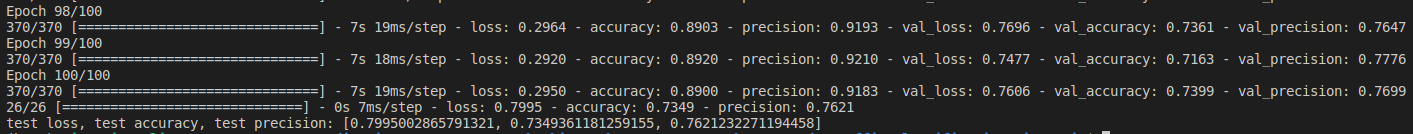


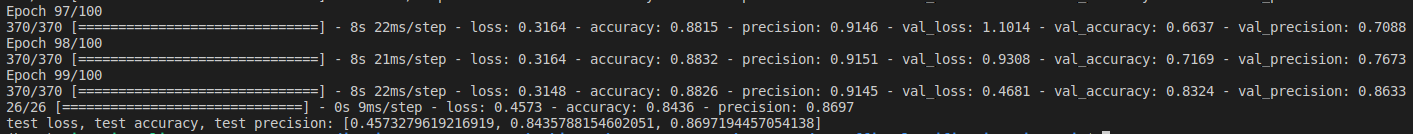
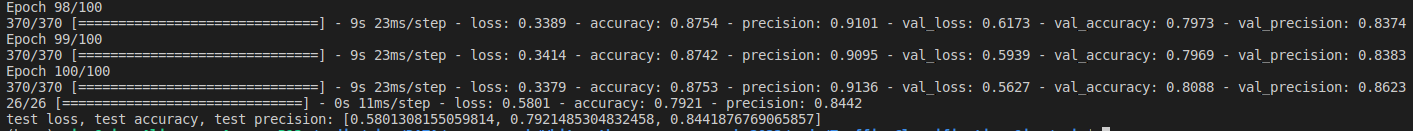
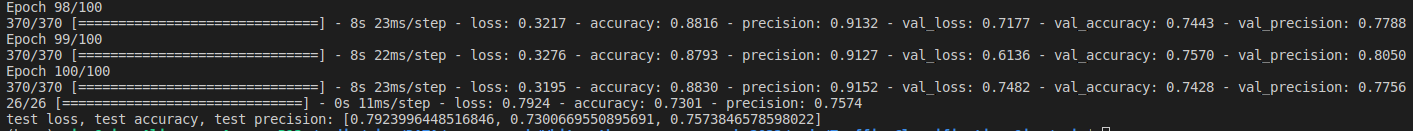
The following 6-layer structure also offers good performance:

MLP: 1200 (0.5) – 980 (0.5) – 640 (0.25) - 460 (0.25) - 220 (0.2) - 80 (0.2)



1. DNN 2.7 (7 hidden layers)
2. MLP: 1600 (0.5) – 1200 (0.5) – 880 (0.25) – 640 (0.25) – 360 (0.25) – 220 (0.2) – 80 (0.2)



1. DNN 2.8 (8 hidden layers)
2. MLP: 1600 (0.5) – 1200 (0.5) – 1200 (0.25) – 880 (0.25) – 640 (0.25) – 360 (0.25) – 220 (0.2) – 80 (0.2)
3. 
4. DNN 2.9 (9 hidden layers)
5. MLP: 1600 (0.5) – 1200 (0.5) – 1200 (0.25) – 880 (0.25) – 640 (0.25) – 640 (0.25) – 360 (0.25) – 220 (0.2) – 80 (0.2)
6. 
7. DNN 2.10 (10 hidden layers)
8. 1600 (0.5) – 1600 (0.5) – 1200 (0.5) – 1200 (0.25) – 880 (0.25) – 640 (0.25) – 640 (0.25) – 360 (0.25) – 220 (0.2) – 80 (0.2)
   1. 

## MLP Scheme 3:

In toTrain.py (line 128), change code to:

X\_train = table\_s3.values

execute (in prepro folder):

python3 toTrain.py Table\_v2.csv

To train the neural network, execute (in main folder):

sh TRAIN.sh DNN

Result:

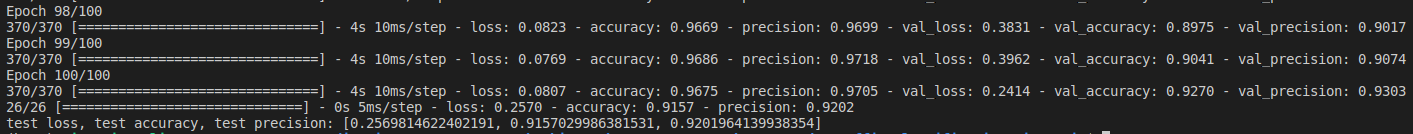


5/16/2023 More data for the paper

In main folder -->train.py, set neural network structure as:

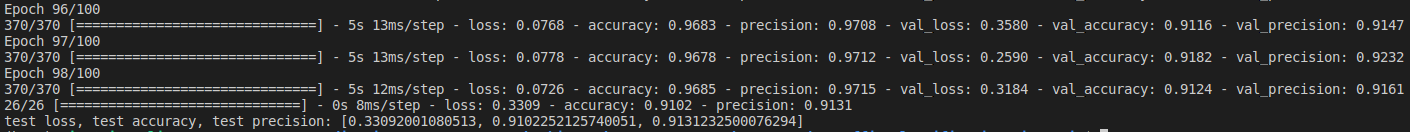
DNN 3.3 (3 hidden layers)

MLP: 1600 (0.5) – 1200 (0.25) – 220 (0.2)



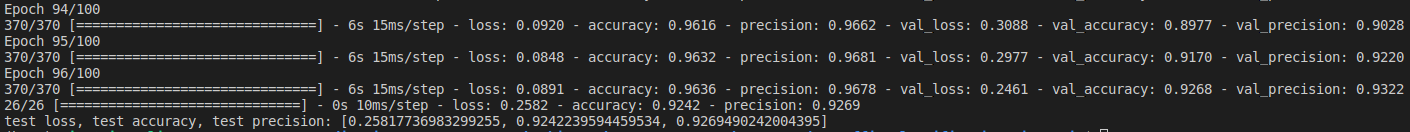
DNN 3.4 (4 hidden layers)

MLP: 1600 (0.5) – 1200 (0.25) – 640 (0.25) – 120 (0.2)



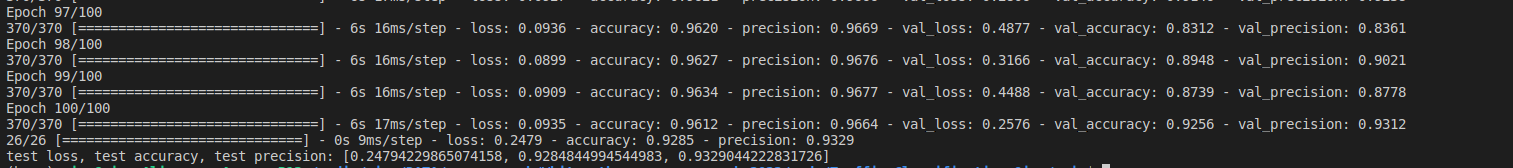
DNN 3.5 (5 hidden layers)

MLP: 1600 (0.5) – 1200 (0.5) – 880 (0.25) – 640 (0.25) – 220 (0.2)



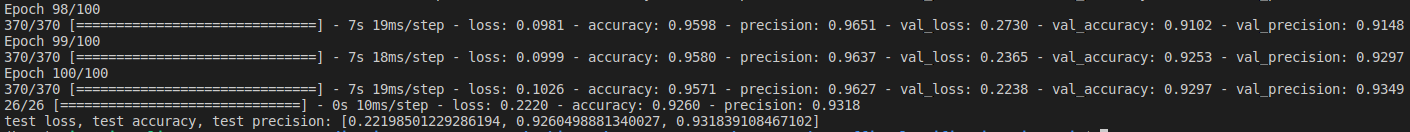
DNN 3.6 (6 hidden layers)

MLP: 1600 (0.5) – 1200 (0.5) – 880 (0.25) – 360 (0.25) – 220 (0.2) – 80 (0.2)

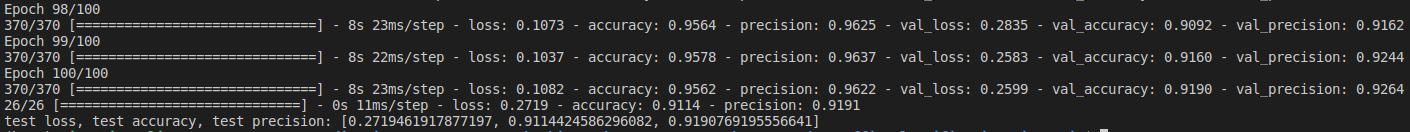
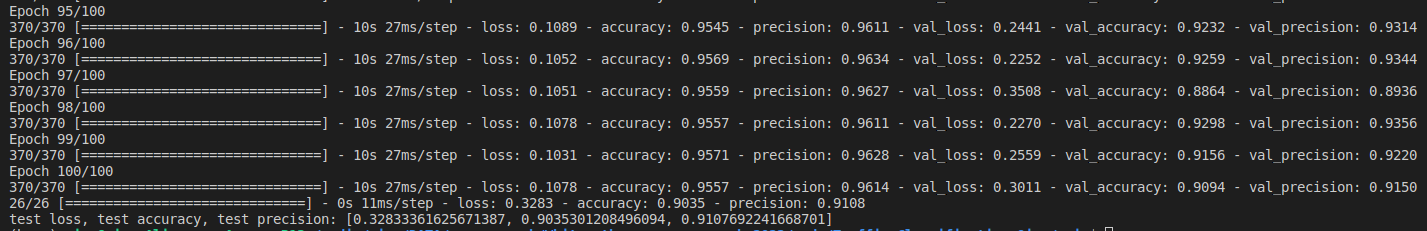


DNN 3.7 (7 hidden layers)

MLP: 1600 (0.5) – 1200 (0.5) – 880 (0.25) – 640 (0.25) – 360 (0.25) – 220 (0.2) – 80 (0.2)

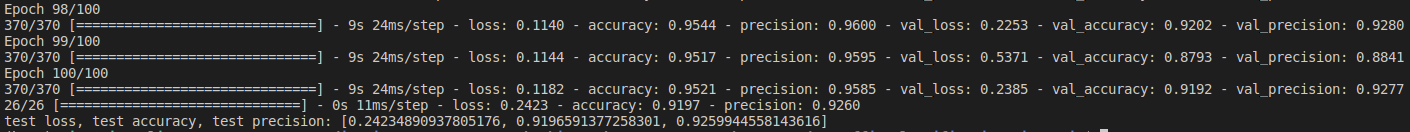


1. DNN 3.8 (8 hidden layers)
2. MLP: 1600 (0.5) – 1200 (0.5) – 1200 (0.25) – 880 (0.25) – 640 (0.25) – 360 (0.25) – 220 (0.2) – 80 (0.2)



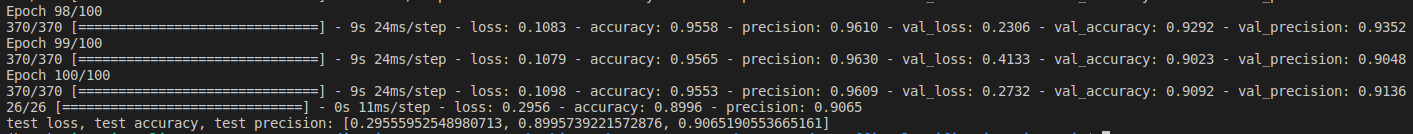
DNN 3.9 (9 hidden layers)

MLP: 1600 (0.5) – 1200 (0.5) – 1200 (0.25) – 880 (0.25) – 640 (0.25) – 640 (0.25) – 360 (0.25) – 220 (0.2) – 80 (0.2)



DNN 3.10 (10 hidden layers)

1600 (0.5) – 1600 (0.5) – 1200 (0.5) – 1200 (0.25) – 880 (0.25) – 640 (0.25) – 640 (0.25) – 360 (0.25) – 220 (0.2) – 80 (0.2)



## CNN Scheme 2:

In toTrain.py (line 128), change code to:

X\_train = table\_s2.values

execute:

python3 toTrain.py Table\_v2.csv

In main folder, set neural network structure as:

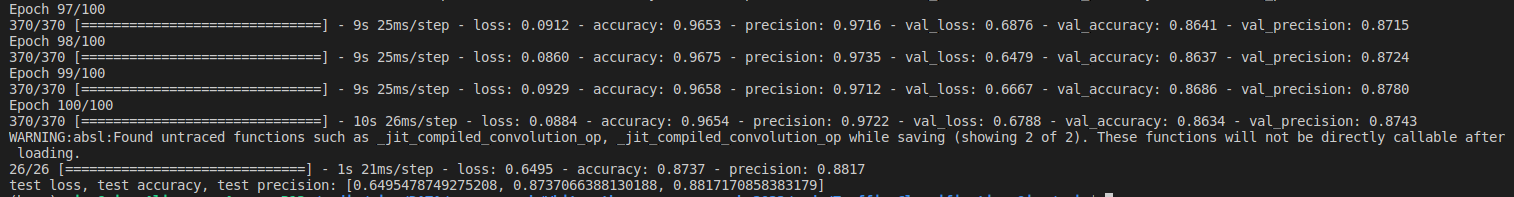
MLP: (100,100) – (100,80) GlobalAveragePooling1D (0.25) – 220 (0.25)

execute:

sh TRAIN.sh DNN

Result:





## CNN Scheme 3:

In toTrain.py (line 128), change code to:

X\_train = table\_s3.values

execute:

python3 toTrain.py Table\_v2.csv

In main folder, set neural network structure as:

execute:

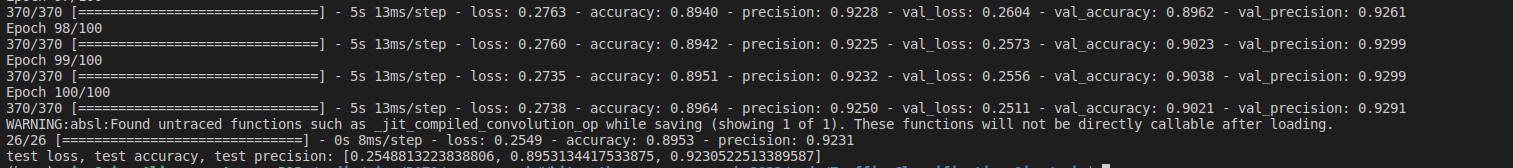
sh TRAIN.sh DNN

Result:



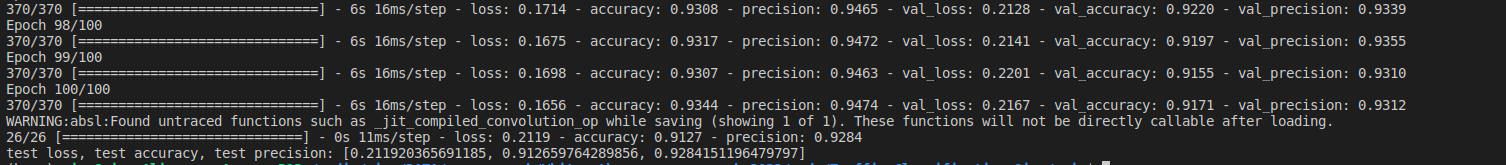
CNN scheme 3.1

CNN: (100,100) – GlobalAveragePooling1D (0.25) – 120 (0.25)



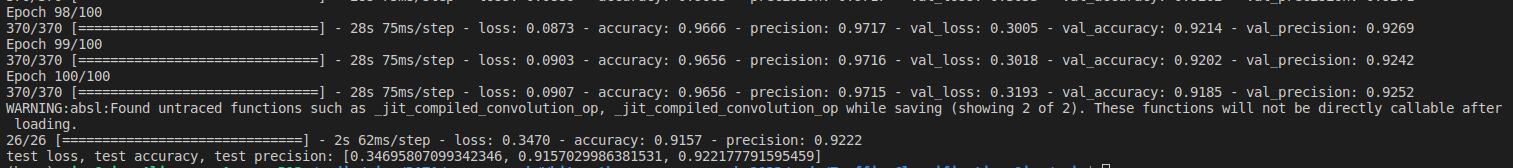
CNN scheme 3.2

CNN: (100,100) – GlobalAveragePooling1D (0.25) – 1200 (0.25) – 580 (0.25) – 120 (0.25)



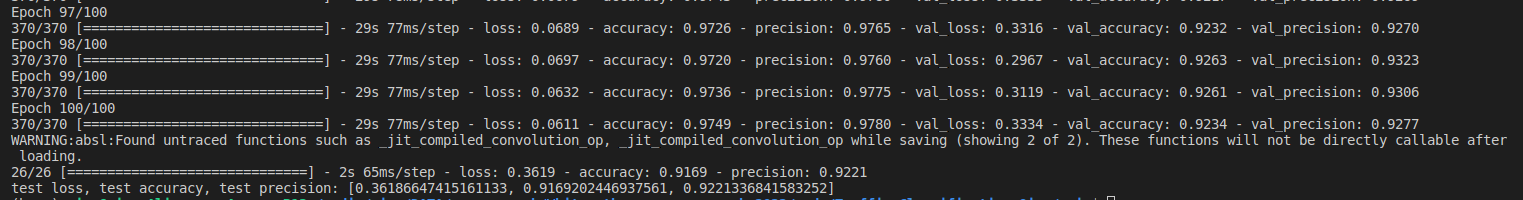
CNN scheme 3.3

CNN: (100,100) – (100,80) – GlobalAveragePooling1D (0.25) – 120 (0.25)



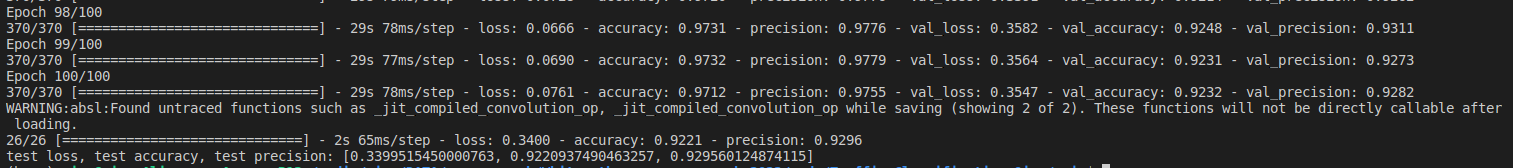
CNN scheme 3.4

CNN: (100,100) – (100,80) – GlobalAveragePooling1D (0.25) – 580 (0.25) – 120 (0.25)

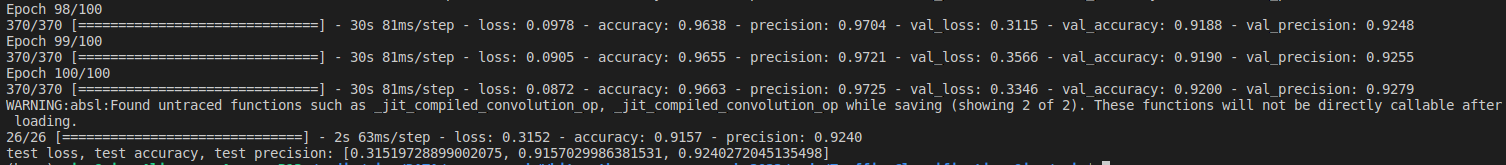


CNN scheme 3.5

CNN: (100,100) – (100,80) – GlobalAveragePooling1D (0.25) – 1200 (0.25) – 580 (0.25) – 120 (0.25)



CNN scheme 3.6

CNN: (100,100) – (100,80) – GlobalAveragePooling1D (0.25) – 1600 (0.25) – 1200 (0.25) – 580 (0.25) – 120 (0.25)

CNN scheme 3.7

CNN: (100,100) – (100,80) – (100,80) – GlobalAveragePooling1D (0.25) – 1200 (0.25) – 580 (0.25) – 120 (0.25)

