Network Intrusion Detection

Kaixuan Hu

1 Anomaly Detection through network tensor and Convolutional Autoencoder

In this section, I will explain the tensor technique we will be using for building a network intrusion detection (IDS) system.

The system we are going to build is based on the unsupervised learning techniques and we only need a minimum amount of prior knowledge and feature engineering on the dataset.

Therefore, we can prevent the consequences of using handcrafting features from human experts based on the dataset, which is introducing biases and affecting the generality of the IDS system. Hence our system can be applied to any other network traffic datasets as long as the dataset contains the essential logging information like the source and destination user, IP address, port number, etc., ordered by the timestamp.

1.1 Traffic Tensor

Viewing the traffic in the form of traffic tensor is based on the work done by Zhang *et. al* [1]

This method periodically captures the state of the network that includes all the information related to the security, flows, authentication events, DNS event, process start, and stops, etc. Each new state is represented by a tensor, i.e. a snapshot from the network. These tensors are arriving continuously.

1.2 Convolutional Autoencoder

The model we are going to apply to the traffic tensors is Convolutional Autoencoder which combines the following two models.

Traffic tensor described above has the same structure as an image. Convolutional neuron network is known for its good performance in image-related applications. By using a small receptive field, CNN can avoid the "fully-connectedness" of the network that prone to overfitting data and can take advantage of the locality of the structure of an image (i.e. neighborhood pixels tend to relate more to each other than the pixels that are far away).

Apart from the Convolutional neuron network, an autoencoder is a type of neural network used to learn efficient data representation (encoding) in an unsupervised manner. It mainly consists of two parts, the encoder and decoder. The encoder part is trained on the data to reduce the dimension of the data (i.e. learn the most compact representation of the data and to ignore noise) while the decoder is trained to reconstruct the encoded data with the objective to minimize the different between the original data and the reconstructed data.

For evaluating the performance of the autoencoder model, we look at the reconstruction error between the original data and the reconstructed data. A small reconstruction error means the autoencoder can generate from the reduced encoding to its original input well.

1.3 The Network Intrusion Detection System

Based on the traffic tensor and Convolutional autoencoder described above, the IDS we are going to build consists of three main parts.

- 1. Generating traffic tensors. There are two sets of traffic tensor, the training set and the testing set.
  - The training set is built from the traffic data that are as 'clean' as possible, in the best-case scenario, the data only contains normal traffic, however, it is often not possible to acquire such traffic data since our goal is to build a IDS on traffic data captured from the real network instead of working with synthetic traffic dataset.

    Acquiring clean training traffic data is one of the challenges for working with real network traffic data, however, as long as the normal background traffic is dominant in the dataset, our autoencoder will have the advantage due to its unsupervised manner, i.e. don't need actual labels for the network data. If it is trained with enough data, it can learn the pattern of the behavior of the normal traffic, once we train our model, we can also use it to evaluate the traffic tensors that are used to train the model by reconstructing them and checking the reconstruction error on those training tensors. We might find some abnormal or malicious traffic lies in the training dataset based on the reconstruction errors (this is one important step for the project for which I will talk more about it in the later section).
  - The testing set can be constructed from the real network traffic or traffic that is injected with some synthetic attack. The purpose of the testing set is after we finished training our model, we want to see if the model is able to identify abnormal traffic from normal traffic.

  To generate the training and testing traffic tensor set, we traverse through the whole traffic table and process it chunks by chunks. Building the training traffic tensor set is one of the most time-consuming parts for building the IDS system, therefore we need to parallelize the process. I use Python `multiprocessing` module to generate tensors in a parallel manner which speeds up the process significantly. However, since the real traffic data tables usually have a very large size, to achieve a good performance using parallelization, it would require a large amount of RAM and as well as a large number of CPU core counts.

  For the testing set, or more generally, for applying our model on new traffic data to detect anomalies, the IDS system with the trained autoencoder can perform the detection in a streaming manner, i.e. given traffic data, the system can construct traffic tensors from the table and feed them into the system to detect any potential anomaly within that traffic data. Unlike the training part, the evaluation part can be done very quickly.

- 2. Train the model on the training set.
- 3. Evaluate the model on the testing set by feeding in the traffic tensors built from the testing set and computing the reconstruction errors for them.

  We need to implement some evaluation methods based on the set of reconstruction errors we get from the traffic tensors (i.e. choose a reasonable threshold for determining normal and abnormal tensor).

If the traffic dataset we are working on has been labeled, then we can use some common metrics like precision and recall and ROC curve to evaluate our model. However, it is often not the case that each record in the data has been clearly labeled, we can use our model to perform the labeling task automatically. How does the automatically labeling works is that, after we train our model using normal traffic data, we can use the model to reconstruct tensors built from other traffic, and based on the reconstruction errors we get, we can isolate traffic corresponding to the anomaly tensors identified by the model. Then we could label them as anomaly traffic. This is one of the potential improvement to the system.

2 Apply to Los Alamos National Laboratory Dataset

2.1 Description of the dataset

The dataset we will be focusing on is Comprehensive, Multi-Source Cyber-Security Events dataset collected from the Los Alamos National Laboratory (LANL) enterprise network. This dataset represents 58 consec-

utive days of de-identified event data collected from five sources within Los Alamos National Laboratory's corporate, internal computer network.

The dataset contains of multiple parts,

- Authentication: **auth.txt.gz**

  This data represents authentication events collected from individual Windows-based desktop computers, servers, and Active Directory servers. Each event is on a separate line in the form of "time, source user@domain, destination user@domain, source computer, destination computer, authentication type, logon type, authentication orientation, success/failure" and represents an authentication event at the given time.

  Here are some examples from the data:

  1, C625$@DOM1, U147@DOM1, C625, C625, Negotiate, Batch, LogOn, Success

  1, C653$@DOM1, SYSTEM@C653, C653, C653, Negotiate, Service, LogOn, Success

  1, C660$@DOM1, SYSTEM@C660, C660, C660, Negotiate, Service, LogOn, Success

- Processes: **proc.txt.gz**

  This data represents process start and stop events collected from individual Windows-based desktop computers and servers.

- DNS: **dns.txt.gz**

  This data presents Domain Name Service (DNS) lookup events collected from the central DNS servers within the network.

- Network Flows: **flows.txt.gz**

  This data presents network flow events collected from central routers within the network.

- Red Team: **redteam.txt.gz**

  This data presents specific events taken from the authentication data that present known redteam compromise events. These may be used as ground truth of bad behavior that is different from a normal user and computer activity. Each event is on a separate line in the form of "time, user@domain, source computer, destination computer" and presents a compromise event at the given time.

  Here are some examples from the data:

  151648, U748@DOM1, C17693, C728

  151993, U6115@DOM1, C17693, C1173

  153792, U636@DOM1, C17693, C294

All data starts with a time epoch of 1 using a time resolution of one second. For security purposes, the specific time frame collected from the operational network is not shown in the dataset. In addition, there are no data that have association outside of the internal enterprise network

We will be focusing one the authentication dataset as well as using the redteam dataset as ground truth for abnormal behavior.

2.2 Dataset preprocessing

Deep Autoencoder Neural Networks for Detecting Lateral Movement in Computer Networks

First, we want to build a custom dataset from the LANL authentication data based on the method described by Holt *et. al* [2].

- Step 1: label the data based on the redteam data by matching the time, user@domain, source computer, destination computer column, if all four column matched in both tables than match the corresponding rows in the authentication dataset as malicious. And at this step, we include the redteam data only.

- Step 2: Add all normal activities performed by the computers associated with each redteam event through source computer or through destination computer, which are labeled as src_comp and dest_comp in the dataset, respectively.

  Therefore, besides the redteam events, new data set also contains all normal events of the computers compromised by at least one redteam event.

  The main reason behind this step is we want to preserve the **lateral movement**, which is a cyber-attack technique that enables an adversary to navigate internal network from machine to machine, specifically, from the initial point of compromise to the objective.

- Step 3: Increase the ratio of noise: randomly choose a set of source computers and add all the events associated with them to dataset (ongoing normal activity in the network).

After the three above, we get a new dataset that is somehow balance comparing to the original data where, the events count of the authentication data is 1,051,430,459 but only 749 of them correspond to redteam events.

The steps on preprocessing the data described below is for building the tensor. They are varied based on different specification of the tensors, here we will only focus on one tensor specification for the experiment.

Step 1: Combine source user at domain column and the source computer column as one column.

Step 2: discard the rows corresponding to the 'Local logon', i.e. rows with the same src_computer and dest_computer. Since those events are not likely to correspond to a Lateral movement (i.e. cyber-attack)

Step 3: focus more on the users that start with U and Anonymous user. (There are three types of source user: Start with C, U or Anonymous user). Another potential point experiment is that focus on the destination computers in the redteam data that later becomes a src_computer in the authentication dataset because victim of a redteam attack might later becomes an attacker.

Selection of the candidate values for src_user_at_computer, dest_computer:

- Source User

  There are three types of source user: Start with C, U or Anonymous user.

  The following table records the count of source user from the data.

```
[57]: important_src_user_value_counts

[57]: ANONYMOUS LOGON@C586    13263955
      U22@DOM1                 7386604
      C585$@DOM1               3400045
      C743$@DOM1               2907729
      C480$@DOM1               2152594
                                ...
      U12231@DOM1                    1
      U8946@C1610                    1
      U12035@C423                    1
      U771@?                         1
      U2470@?                        1
      Name: 0, Length: 30482, dtype: int64
```

- Source User at Source Computer

  We combine the source user column and the source computer column as a new column which will be one of the dimension of our tensor. The main reason for this combination is that since our tensor only has three dimension (increasing the dimension will require more amputating resource, for the purpose of experimenting the method, we don't want to put more dimensions in our tensor), by combing the two column, we allow the tensor to keep more information while not increase the dimension of the

tensor and we can also take into account the connection between the source user column and the source computer column.

The following table records the count of source user at source computer from the data.

```
[62]: src_userATsrc_comp_value_count

[62]: C585$@DOM1@C585            3400045
      C743$@DOM1@C743            2907729
      C480$@DOM1@C480            2152594
      C599$@DOM1@C1619           1866002
      U22@DOM1@C506              1858072
                                  ...
      U1653@DOM1@C25345                1
      U11406@DOM1@C1835                1
      U8120@?@C17526                   1
      U9980@DOM1@C1139                 1
      U11624@lanl.gov@C12183           1
      Name: 0, Length: 124850, dtype: int64
```

- Destination computer

  The following table records the count of destination user from the data.

```
[65]: pd.read_csv("May14Data/full_dest_comps_value_counts.csv")
```

[65]:

	dest_computer	count
0	C586	65984736
1	C612	37616163
2	C457	36690585
3	C529	36350917
4	C467	36042520
...	...	...
723	C8422	1
724	C501	1
725	C3147	1
726	C3668	1
727	C9226	1

728 rows × 2 columns

2.3 Tensor specification

The tensor is designed to have three dimension: [`src_user_at_computer`, `dest_computer`, `time_window`]. We decided to select 400 src_user_at_computer and 400 dest_comp and the time_window is [5 sec, 60 sec, 3600 sec].

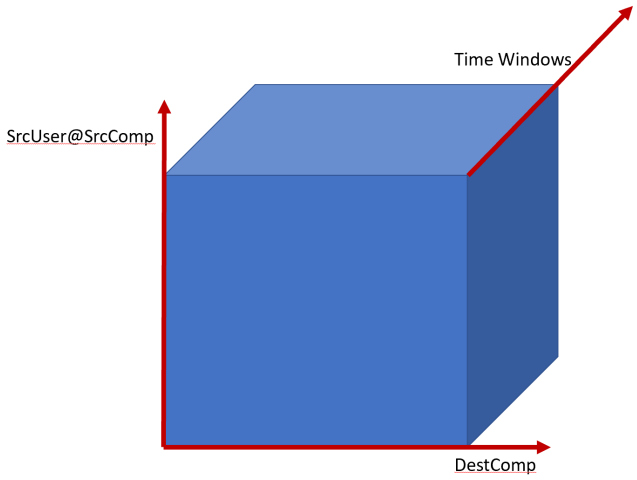
Ideally, we should consider all the user and computer in the dataset, but, due to the size of the dataset, and for a purpose of the experiment we will only focus on parts of the user and computer. We select the candidate users and computers based on the counting tables shown above, which are the top 400 values from the tables.

The dimension of the tensor is $400 \times 400 \times 3$. The reasoning behind the design of the tensor: we pick `src_user_at_computer`, `dest_computer` as the first two dimensions for the tensor is we want the tensor be

able to preserve the lateral movement in the dataset (by the similar reason as the way we preprocess the data and build a custom dataset).

We can use the tensor to perform a query that given a value for src_user_at_computer, a destination computer and a time window, we can get the count of occurrence of rows from the original data.

The tensor will have the following shape:



2.4 Future work

I haven't build a deep learning model for the tensor from LANL, because after working with the data and getting familiar with it, we think building a model on the data for achieving a good result might be challenging comparing to the other datasets in the same field (e.g. UGR) since the characteristic of the lateral movement in the data is not very distinct comparing to the normal traffic, therefore, we decide to do experiments on the UGR data using the same tensor technique and the convolution autoencoder to see if the method can achieve some good performance on identifying anomaly traffic. After that, we might go back to the LANL data and apply the same method to build a network intrusion detection system (IDS).

It is also worth mentioning an advantage of our method is that it is transferable to different network traffic datasets as long as the dataset contains the essential logging information like the source and destination user, IP address, port number, etc.

3 Apply to UGR'16

In this section, I will present experiments applying the tensor technique to the UGR'16 data.

3.1 Description of the dataset

UGR'16 data come from several netflow v9 collectors located in the network of a Spanish ISP. It is composed of two sets of data that are previously split in weeks:

- A CALIBRATION set of data gathered from March to June of 2016 (4 months) containing real background traffic data.
- A TEST set of data gathered from July to August of 2016 containing real background and synthetically generated traffic data that corresponds with several and well known types of attacks.

For the experiments, I will use June week 1 data from the calibration dataset and July week 5 data from the test dataset.

3.2 Tensor specification

For working with the UGR'16 dataset, we have done multiple experiments on different tensor specifications. And we have settled down to the tensor specification that we think it is the most reasonable compared to the other.

3.2.1 Specification #1

First, the tensor has three dimensions, unlike the tensor specification for the LANL's dataset. The three dimensions are designed to be: [`src_IP`, `dest_IP`, `dest_port`]. We replace the dimension for the time window with the destination port dimension and fix the time window for all tensors (e.g. each tensor is generated based on 60 seconds duration of traffic). By the replacement, we can let the model take into account more aspects of the traffic, and destination port certainly contains more information about the traffic than the time window, and it is more worthwhile to use destination port. For this experiment, the shape of the tensor is $40 \times 40 \times 20$.

For the source and destination IP, we select the part of the IP addresses that we think are important. Specifically, the original UGR'16 paper [3] mentioned 20 Victim IP addresses and 5 attacker IP addresses. To catch the malicious behaviour from the traffic, those IP addresses that are explicitly marked should be monitored.

Machine/s	**IP Address/es**
A_1 - A_5	42.219.150.{*246,247,243,242,241*}
V_{11} - V_{15}	42.219.156.{*30,31,29,28,27*}
V_{21} - V_{25}	42.219.158.{*16,17,18,19,21*}
V_{31} - V_{35}	42.219.152.{*20,21,22,23,18*}
V_{41} - V_{45}	42.219.154.{*69,68,70,71,66*}

Only selecting one particular IP as one value in our tensor is very inefficient since the traffic size is large and there are millions of different IP addresses within the network. Therefore, instead of considering one IP address, we use subnets, i.e. we consider groups of IP addresses as one value in our tensor so that we can let the tensor represent more information from the data.

For the network where the UGR'16 is captured and built, the subnet 42.219 is more important than the other, therefore, we select 13 important subnets that start with 42.219 to be the candidate value. Right now we already have $20 + 5 + 13 = 38$ candidate values for the IP addresses, the remaining values are i) all the traffic to or from subnet 42.219, ii) all the traffic to or from subnets other than 42.219. Note that for the last two values, each of them counts toward one condition for the candidate source and destination IP, for example, we count all the rows satisfying they all come from IP addresses other than subnet 42.219 together even if they may come from the different IP address (src_IP).

The following shows the 20 destination ports we select

- 80 (TCP), 53 (TCP/UDP), 443 (TCP), 445 (TCP),
- 25 (TCP), 23 (TCP), 110 (TCP), 123 (UDP),
- 22 (TCP), 21 (TCP), 143 (TCP), 161 (UDP)
- 6 port ranges:
  - all other ports below 1024;
  - 1024 - 2047; 2048 - 4097; 4098 - 16391, 16392 - 49151; 49152 - 65535
- All the ports with protocol UDP (excluding 53, 123, 161)
- All the ports with protocol ICMP

There are many drawbacks to this specification. First, after we build the tensors, we found that the 38 important source and destination IP addresses/subnets don't communicate with each very often, instead, they have lots of traffic going to/coming from the subnets other than 42.219, meaning that the most of the cells in the tensors contain zero, and the cells correspond to: i) all the traffic to or from subnet 42.219, ii) all the traffic to or from subnets other than 42.219 contain a very large count. Next, the selection of the destination port ranges is not even, we should change it to intervals that have equal size so that the model built upon it does not bias towards some of the intervals.

3.2.2 Specification #2

In this section, I will show the updated tensor specification and some experiment results.

The three dimension is designed to be: [`src_IP`, `dest_IP`, `dest_port`] with shape $64 \times 64 \times 64$.

Selection of IP addresses:

```
ip_lst = [
    '143.72', '43.164', '204.97', '168.38',
    '194.233', '106.150', '210.46', '70.211',
    '133.54', '78.160', '74.158', '54.143',
    '74.159', '214.43', '165.129', '209.48',
    '42.219.159', '42.219.158', '42.219.157', '42.219.156',
    '42.219.155', '42.219.154', '42.219.153', '42.219.152',
    '42.219.151', '42.219.150', '42.219.149', '42.219.148',
    '42.219.147', '42.219.146', '42.219.145', '42.219.144',
    (0, 7), (8, 15), (16, 23), (24, 31), (32, 39), (40, 47),
    (48, 55), (56, 63), (64, 71), (72, 79), (80, 87), (88, 95),
    (96, 103), (104, 111), (112, 119), (120, 127), (128, 135),
    (136, 143), (144, 151), (152, 159), (160, 167), (168, 175),
    (176, 183), (184, 191), (192, 199), (200, 207), (208, 215),
    (216, 223),(224, 231),(232, 239), (240, 247),(248, 255)
]
```

The IP candidate list consists of three parts,
i) 16 most frequently appear x1.x2 subnets within the whole calibration data.
ii) 16 most frequently appear x1.x2.x3 subnets within the whole calibration data.
iii) 32 IP address ranges, goes from 0.0.0.0 - 7.255.255.255, 8.0.0.0 - 15.255.255.255 to . . . 255.255.255.255

Selection of destination port and protocol:

```
(0, 127), (128, 255),
(256, 511), (512, 767),
(768, 1024), (1024, 2047),
(2048, 3071), (3072, 4095),
(4096, 5119), (5120, 6143),
(6144, 7167), (7168, 8191),
(8192, 9215), (9216, 10239),
(10240, 16383), (16384, 22527),
(22528, 28671), (28672, 34815),
(34816, 40959), (40960, 47103),
(47104, 53247), (53248, 59391),
(59392, 65535),
```

```
(53, ('TCP', 'UDP')), (80, 'TCP'), (80, 'UDP'),
((81, 8080, 8081, 8888, 9080, 3128, 6588, 7779, 1080), ('TCP', 'UDP')),
(443, ('TCP', 'UDP')), (445, ('TCP', 'UDP')),
((25, ('TCP', 'UDP')), (587, 'TCP')),
((23, 2323, 9527), ('TCP', 'UDP')),
(((20, 21), ('TCP', 'UDP')), (69, 'UDP')), (8000, 'TCP'),
((110, ('TCP', 'UDP')), (995, 'TCP')),
((161, 162), ('TCP', 'UDP')),
(22, ('TCP', 'UDP')),
(123, ('TCP', 'UDP')),
((5060, 5061), ('TCP', 'UDP')),
((143, 993), ('TCP', 'UDP')),
(389, ('TCP', 'UDP')),
((3306, ('TCP', 'UDP')), (1433, 'TCP')),
((768,769,770,771,
  772,773,774,775,
  776,777,778,779,
  780,781,782,783),
 'ICMP'),
((0, 2048), 'ICMP'),
((1720, ('TCP', 'UDP')), ((1002, 5222), 'TCP')),
((3389, 'TCP'), ((5900, 5910), 'TCP'), ((5800, 5810), 'TCP'), (5500, 'TCP')),
((135, 136, 137, 138, 139), ('TCP', 'UDP')),
(((500, 4500), 'UDP'), ((1701, 1723), ('TCP', 'UDP')), (0, 'ESP'), (0, 'AH')),
((1900, ('TCP', 'UDP')),
 ((6, 5000, 5431, 2048, 2869, 1900, 5351, 37215, 18067), 'TCP')),
(1723, 'TCP'),
6667, # (0, 'GRE'),7/13 updated
(((6881, 6999), ('TCP', 'UDP')), ((27000, 27050), ('TCP', 'UDP'))),
(((111, 135), ('TCP', 'UDP')), ((6000, 6063), 'UDP')),
((554, 7070, 9090, 22010), ('TCP', 'UDP')),
((1025, 1029), ('TCP', 'UDP')),
(((6343, 8291, 8728, 8729, 4153), 'TCP'), ((5678, 20561), 'UDP')),
(520, 'UDP'),
(((5938, 55555, 6379), 'TCP'), ((3383, 1233), 'UDP')),
((5222, 5228), 'TCP'),
((32764, 'TCP'), ((53413, 39889), 'UDP')),
((5555, 7547, 30005), 'TCP'),
((9100, 515, 631, 81, 10554), 'TCP'),
(((8083, 5678), ('TCP', 'UDP')),
 ((8181, 4786, 8443, 8007, 8008, 8009, 23455, 5380, 4567), 'TCP'),
 (18999, 'UDP')),
(((61001, 37215, 52869, 2000, 7676), 'TCP'), (9999, ('TCP', 'UDP'))),
(((10000, 4444, 27374, 1050), ('TCP', 'UDP')), (1024, 'TCP'))]
```

For destination port value, there are 64 in total, divided into 2 parts.

i) 23 port ranges:

$$(0, 127), (128, 255), (256, 511), (512, 767), \ldots, (53248, 59391), (59392, 65535 (= 2^{16} - 1))$$

ii) The remaining 41 values are the port and protocol combinations that correspond to different services. For instance, TCP/UDP:53 corresponds to DNS, TCP/UDP:443 corresponds to SSL, TCP/UDP:22 corresponds to SSH, TCP:5222/5228 is used by the Google Playstore.

Comparing to the previous tensor specification, this one avoids the use of a single IP address which is very inefficient. And apart from the important value for IP address and port/protocol combination, we divide the interval for them evenly to avoid introducing biases in the tensor and the model.

The following shows the structure of the neuron network model

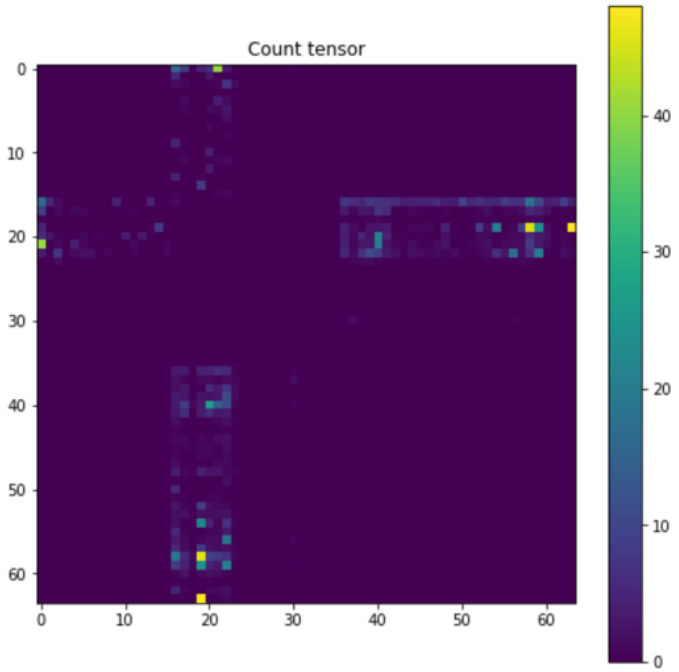
```
class Autoencoder(nn.Module):
    def __init__(self):
        super(Autoencoder, self).__init__()
        self.encoder = nn.Sequential(
            nn.Conv3d(1, 32, 3, stride=2, padding=1),
            nn.ReLU(), # or SELU()
            nn.Conv3d(32, 64, 3, stride=2, padding=1),
            nn.ReLU(),
            nn.Conv3d(64, 128, 3, stride=2, padding=1),
            nn.ReLU(),
            nn.Conv3d(128, 256, 3, stride=2, padding=1),
            nn.ReLU()
        )
        self.decoder = nn.Sequential(
            nn.ConvTranspose3d(256, 128, 3, stride=2, padding=1, output_padding=1),
            nn.ReLU(),
            nn.ConvTranspose3d(128, 64, 3, stride=2, padding=1, output_padding=1),
            nn.ReLU(),
            nn.ConvTranspose3d(64, 32, 3, stride=2, padding=1, output_padding=1),
            nn.ReLU(),
            nn.ConvTranspose3d(32, 1, 3, stride=2, padding=1, output_padding=1),
            nn.ReLU()
        )

    def forward(self, x):
        x = self.encoder(x)
        x = self.decoder(x)
        return x
```

The model consists of one encoder and one decoder, each has four 3D convolution layers, the tensor dimension will be reduced four times to the latent dimension $4 \times 4 \times 4$ by the encoder and the decoder is the opposite of this operation. The following shows the change of the tensor shape when going through the encoder:

$$64 \times 64 \times 64 \to 32 \times 32 \times 32 \to 16 \times 16 \times 16 \to 8 \times 8 \times 8 \to 4 \times 4 \times 4$$

Taking the average across the destination dimension, we can visualize the tensor:



3.3 Experiment results

Using the tensor specification #2, I did the following experiments. For the training set, I used June week 1 data and keep only the traffic records labeled as background. For the testing set, I used July week 5 data. Each tensor has a starting time and a duration of 60 seconds. To label the tensor, I used the provided attack execution timestamp table. The tensor labeling process is based on the starting time of the tensor, note that the attack execution timestamp has a resolution of 1 minute (see the following image), if the starting time of a tensor lies within some 1-minute interval of the table, then the tensor is labeled according to that. It is worth noting that this is not the most precise way for labeling since there are some tensors whose main part does not lie within the 1-minute interval they are assigned. Better ways of labeling could be defined, but for the purpose of this experiment, it is fine to have a small deviation.

	counter(mins)	Dos	scan44	scan11	nerisbotnet	blacklist	anomaly-udpscan	anomaly-sshscan	anomaly-spam	attack
2016-07-27 14:10:00	32	0	0	0	0	1	0	0	1	1
2016-07-27 14:11:00	33	0	0	0	0	1	0	0	1	1
2016-07-27 14:15:00	37	0	0	0	0	1	0	0	1	1
2016-07-27 14:16:00	38	0	0	0	0	1	0	0	1	1
2016-07-27 14:18:00	40	0	0	0	0	1	0	0	1	1
...	...	...	...	...	...	...	...	...	...	...
2016-07-31 19:29:00	6111	1	0	0	0	1	0	0	0	1
2016-07-31 19:30:00	6112	1	0	0	0	1	0	0	0	1
2016-07-31 19:31:00	6113	1	0	0	0	1	0	0	0	1
2016-07-31 19:32:00	6114	1	0	0	0	1	0	0	0	1
2016-07-31 20:31:00	6173	0	0	0	0	1	0	1	0	1

1364 rows × 10 columns

3.3.1 Identify labelled attack by reconstruction errors

After the model is trained on the training set, I plot the tensor reconstruction error on the training set and the testing set.

Figure 1: Training tensors reconstruction errors

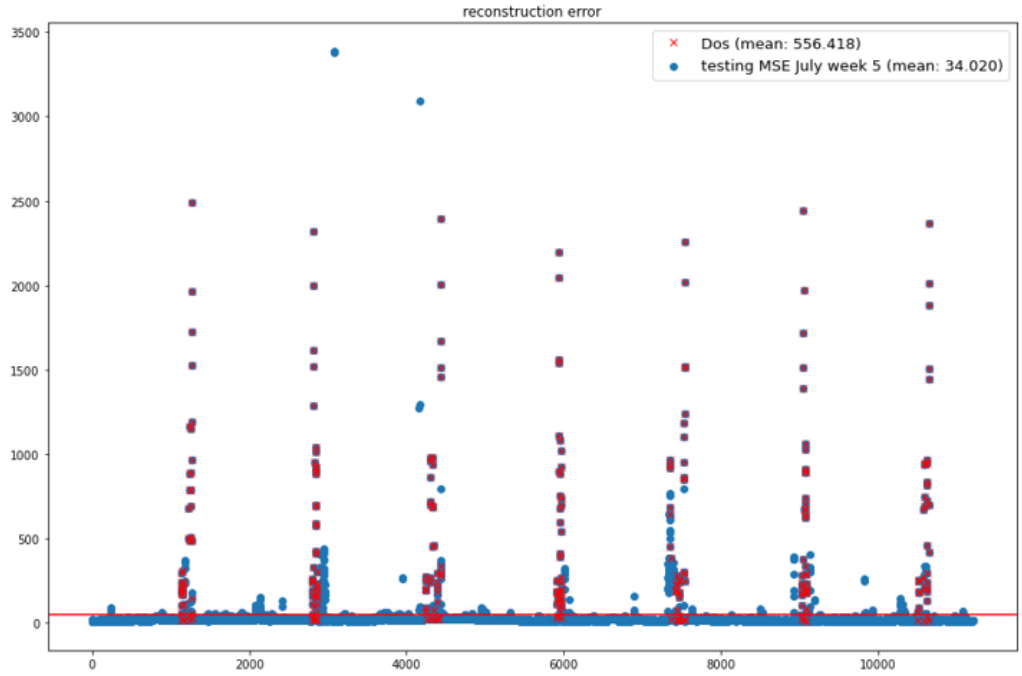


Figure 2: Testing tensor reconstruction errors. The tensors labelled as Dos are marked by red cross, can identify 129 Dos out of 136

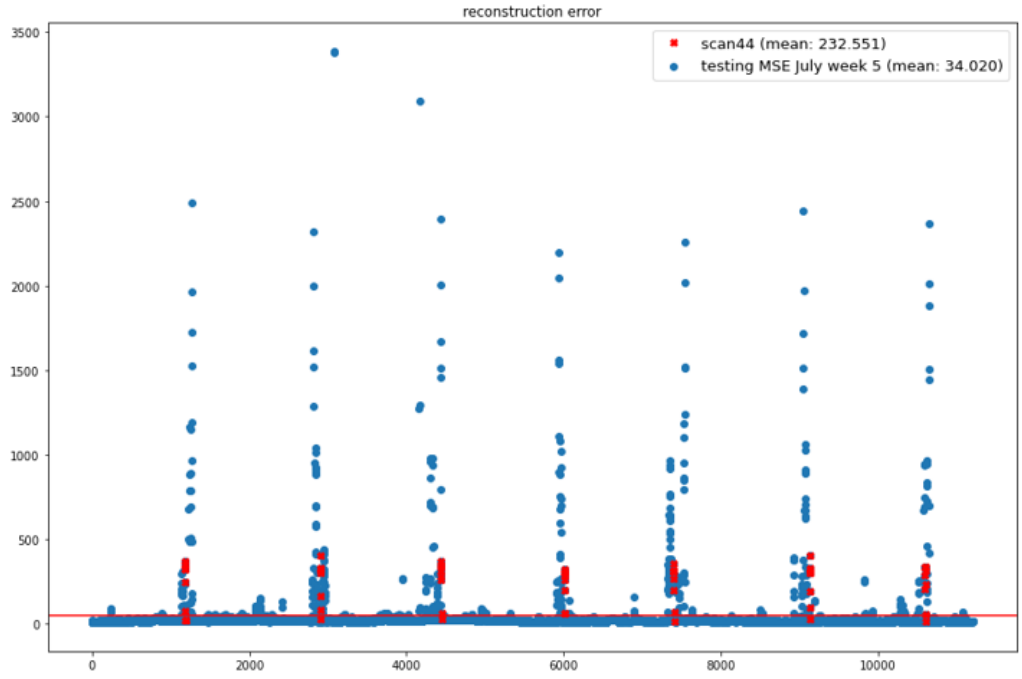


Figure 3: Testing tensor reconstruction errors. The tensors labelled as scan44 are marked by red cross, can identify 27 scan44 out of 28

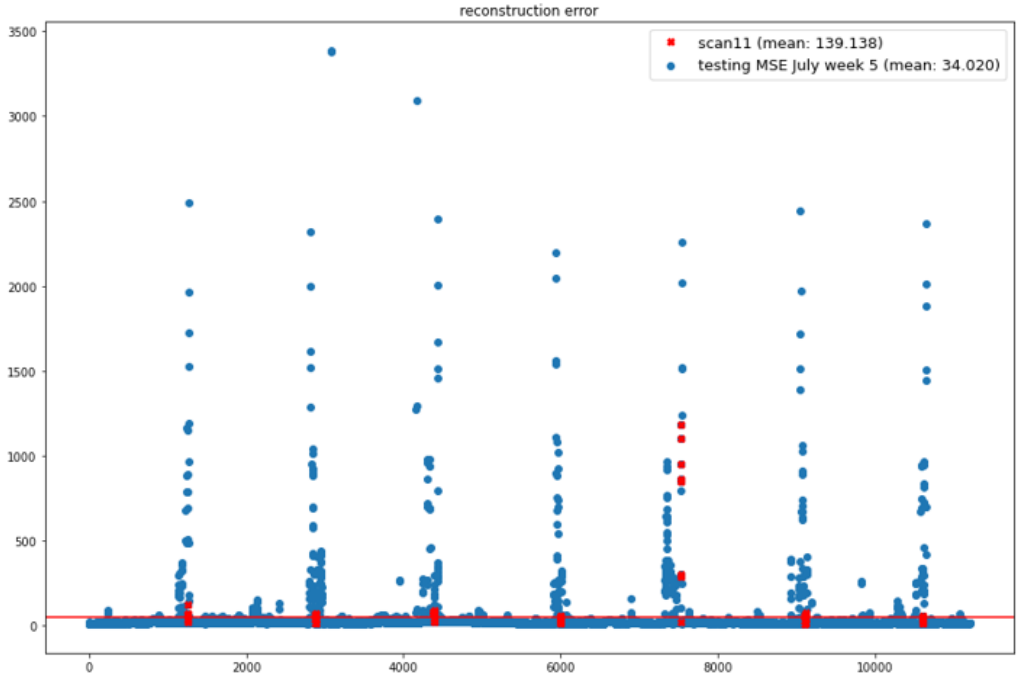


Figure 4: Testing tensor reconstruction errors. The tensors labelled as scan11 are marked by red cross, can identify 20 scan11 out of 28

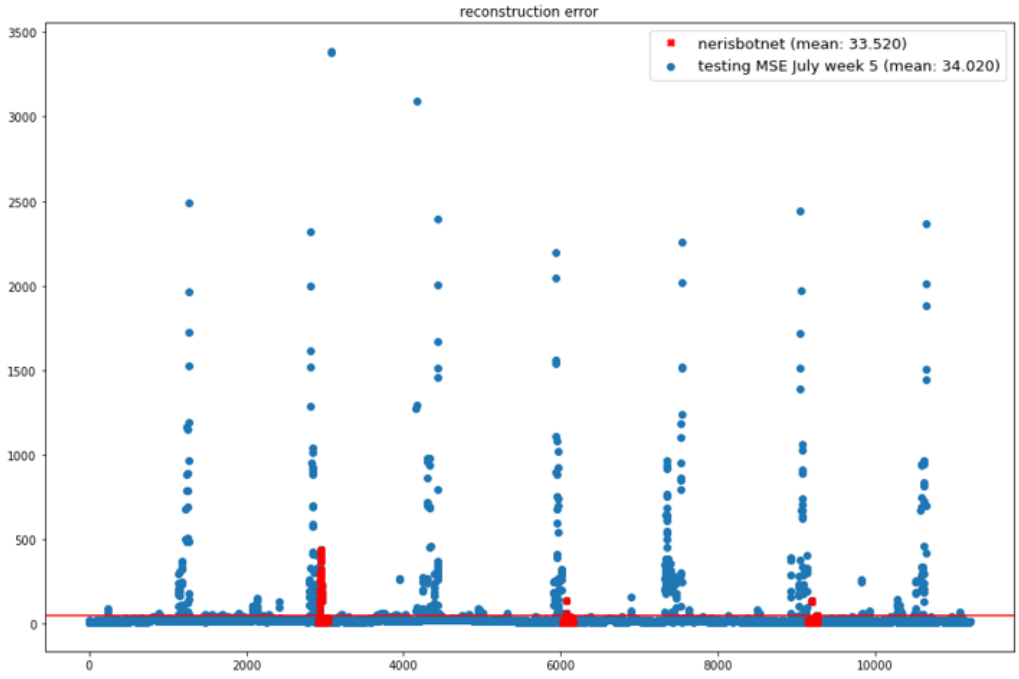


Figure 5: Testing tensor reconstruction errors. The model can identify 17 nerisbotnet out of 183

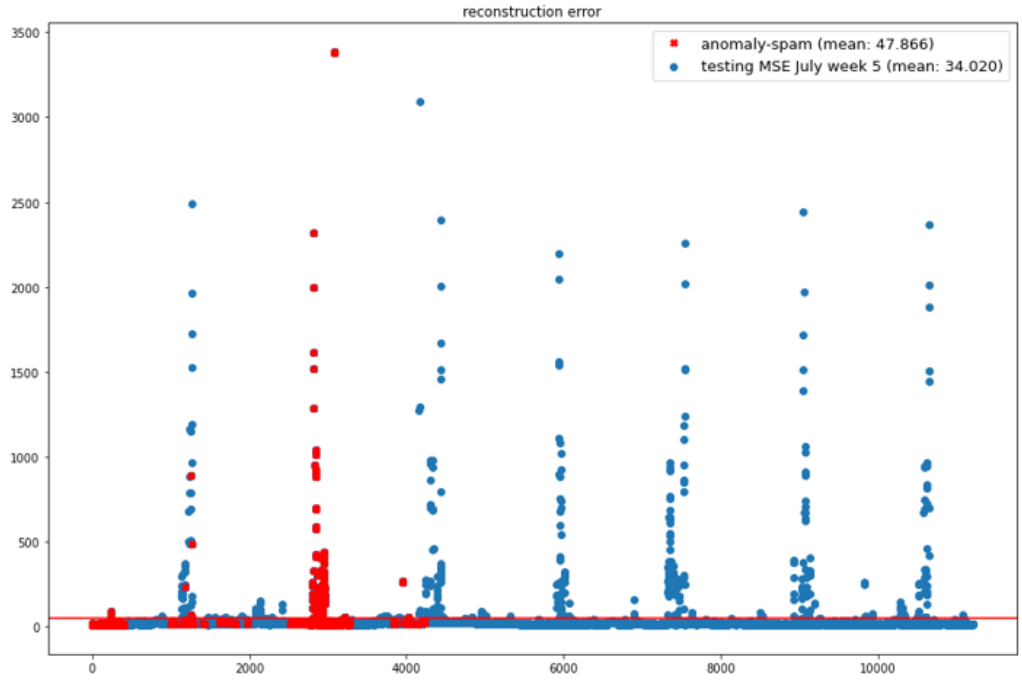


Figure 6: Testing tensor reconstruction errors. The model can identify 51 anomaly-spam out of 609

Comments:

- The x-axis is the tensor indices sorted by timestamp. The y-axis is the tensor reconstruction error.
- The red lines on each plot have value equals to 50, i.e. the reconstruction error threshold for distinguishing between normal and abnormal tensor is 50.
- The number on the top right of each plot is the average reconstruction error for all the tensors, the training tensors have the lowest overall reconstruction error, while the testing tensors has a higher average reconstruction error. The labeled attacks have the highest values.
- More description on the prediction rate I marked on each plot: as I mentioned above, each tensor has a starting time, and the attack execution table is divided by the 1-minute interval. The rate is computed by grouping the tensor into the 1-minute interval and get the corresponding number, for example, 129 Dos out of 136 has a reconstruction error greater than 50, 136 is the number of minutes that has Dos label, 129 is the number of tensors has Dos label and its reconstruction error is greater than the threshold we define (50 in this case).
- Fig. 2 to 6 are all the same plots but with only different labels. We can observe a periodic pattern for the increase of the reconstruction error (i.e. the spike on the plots), this is as expected since according to the original UGR'16 paper [3], they execute attack periodically based on the table shown below.
- We can see from the prediction rate, the model perform well on identifying Dos, scan11, scan44, but perform poorly on identifying nerisbotnet and anomaly-spam. One reason is that Dos, scan11/44 attacks usually generate a large volume of traffic within a short period of time, therefore, the corresponding tensors generated will look very different from a normal tensor, and based on the simple rule of the reconstruction error, we can easily identify those attacks. However, the model does not has a good performance in identifying nerisbotnet and anomaly-spam, the next experiment shows the effort on improving the model in identifying those attacks.

TABLE II
PLANNED SCHEDULING FOR EACH OF THE ATTACKS WITHIN A TIME INTERVAL OF 2H STARTING AT t_0.

Starting time	Attack	Duration
$t_0 + 0h00m$	DoS11	3m
$t_0 + 0h10m$	DoS53s	3m
$t_0 + 0h20m$	DoS53a	10m
$t_0 + 0h40m$	Scan11	3m
$t_0 + 0h50m$	Scan44	3m
$t_0 + 1h00m$	Botnet	1h

TABLE III
DATE AND TIME FOR THE EXECUTION OF THE DIFFERENT ATTACK BATCHES IN THE DATASET.

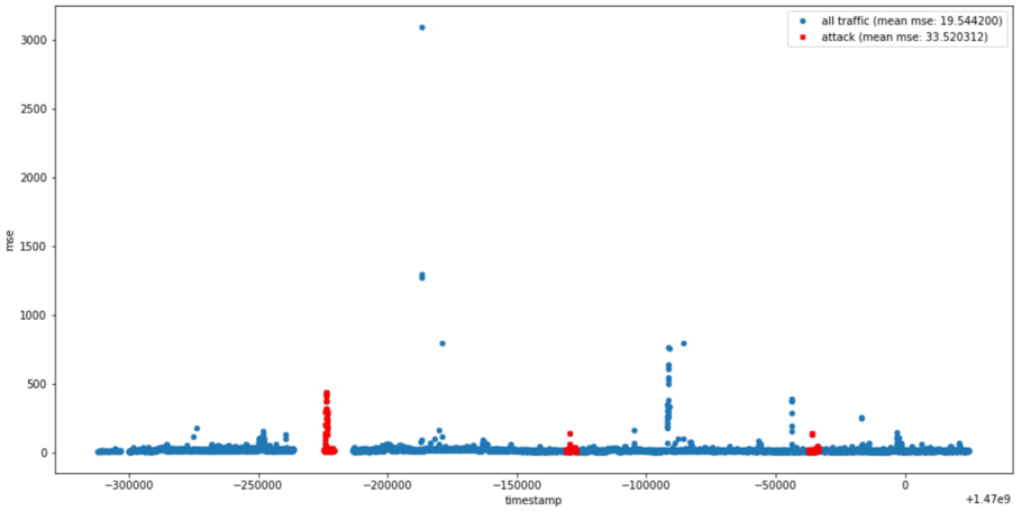
Date	Planned	Random
Thu, 07/28/2016	00:00	12:00
Fri, 07/29/2016	02:00	14:00
Sat, 07/30/2016	04:00	16:00
Sun, 07/31/2016	06:00	18:00
Mon, 08/01/2016	08:00	20:00
Tue, 08/02/2016	10:00	22:00
Wed, 08/03/2016	12:00	N/A
Thu, 08/04/2016	14:00	00:00
Fri, 08/05/2016	16:00	02:00
Sat, 08/06/2016	18:00	04:00
Sun, 08/07/2016	20:00	06:00
Mon, 08/08/2016	22:00	08:00
Tue, 08/09/2016	N/A	10:00

3.3.2 Improving the model in identifying nerisbotnet

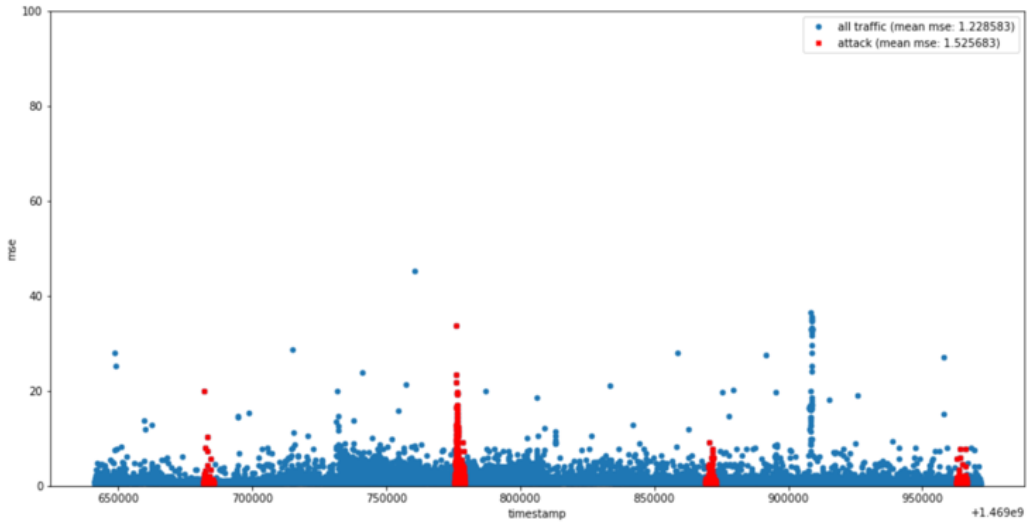
- Normalization

  The normalization method I use is change to count to percentage for each cells in the tensor. $x' = x/sum(\mathbf{x})$. For comparison for the model with or without normalization in identifying nerisbotnet, I plot the confusion matrix. The reconstruction error threshold for both models is 1.2 times the average testing reconstruction error.

  The model without normalization produced the following reconstruction error plot, using a threshold of 1.2 times the average testing reconstruction error, we get the recall on nerisbotnet=0.21, precision=0.05



The model with normalization produced the following reconstruction error plot, using a threshold of 1.2 times the average testing reconstruction error, we get the recall on nerisbotnet=0.43, precision=0.09



Comments:

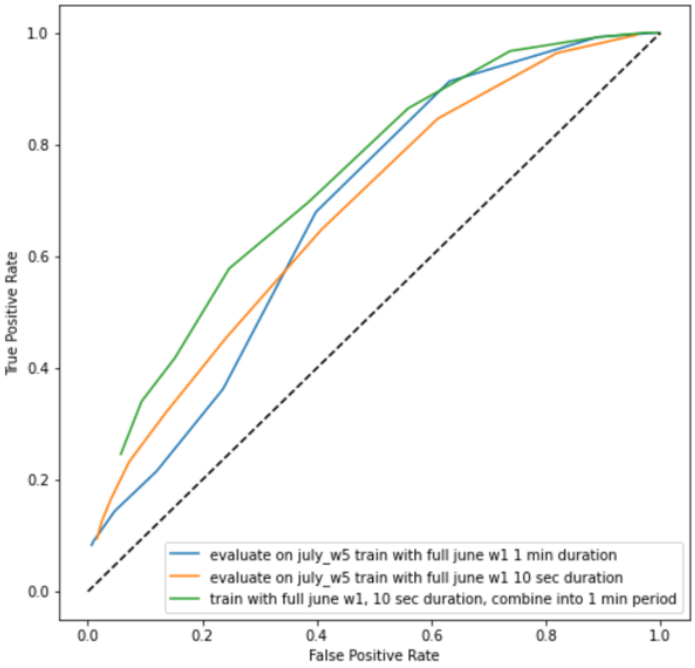
i) The new experiment contains more data from July week 5 (the first red points group in the second plot above is the new data).

ii) From both plots, the results in identifying botnet are not good (indicating change in tensor specification or model is needed), but we know normalization can help to improve the result from this experiment since by normalizing the tensor from count to percentage, we can avoid the effect of the different volumes of the traffic for different periods of time.

- Shorten the tensor duration

  Shorten the tensor duration from 1 minute to 10 seconds will improve the result. I plot the ROC curve for comparing both models. The ROC curve is generated by plotting different values of the thresholds and get different true positive rates (TPR) and false positive rates (FPR), i.e. we use the reconstruction error plots for both duration's model and around the horizontal threshold line to get a list of prediction TPR, FPR. We should focus on the green curve and the blue curve which correspond to the nerisbotnet prediction ROC curves for 10 seconds and 1 minute respectively. NOTE: the orange curve is for 10 seconds model but the evaluation based on 10 seconds duration, we combine all tensors within the same 1-minute interval. As long as one of the tensors within the 1 minute has a positive

prediction, then that 1 minute is count as one positive prediction.

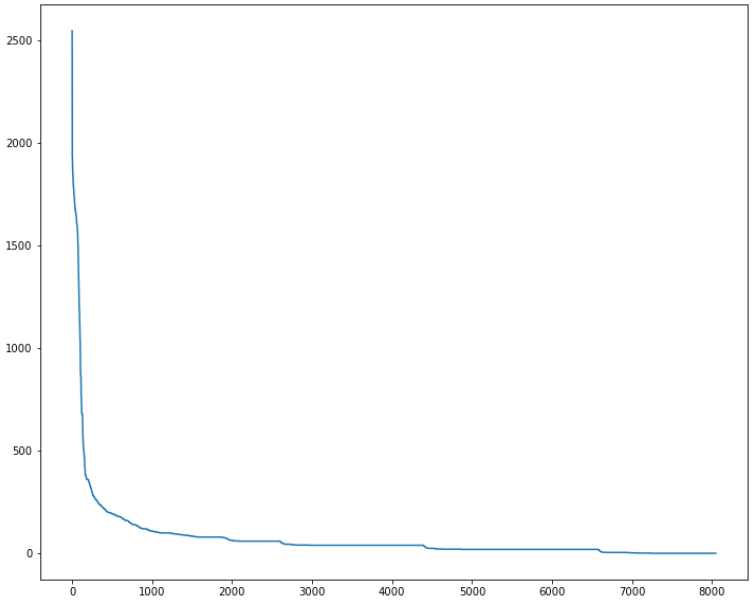


From the plot, we can see that shorten the duration of the tensor, the model can perform better in identifying nerisbotnet, this matches our expectation since nerisbotnet attack does not increase the traffic volume dramatically comparing to Dos or scan attack, but they are concentrated, only a small portion (about 500 seconds) of about 8000 seconds traffic labeled as nerisbotnet creates a significant anomaly. (See the following plots).

Therefore, if we reduce the time interval for each tensor, the spikes caused by the botnet will be more visible (anomalous) in the corresponding tensor

The following plot shows that the botnet traffic is concentrated on a small portion of all the seconds. The x-axis refers to the second, there are about 8000 seconds in July week 5 have some botnet traffic, the y-axis refers to the count of the botnet records given the timestamp x (each second has more than one records of the traffic in the UGR'16 data table). Sort the time value based on the count (y value),

we get the following plot.

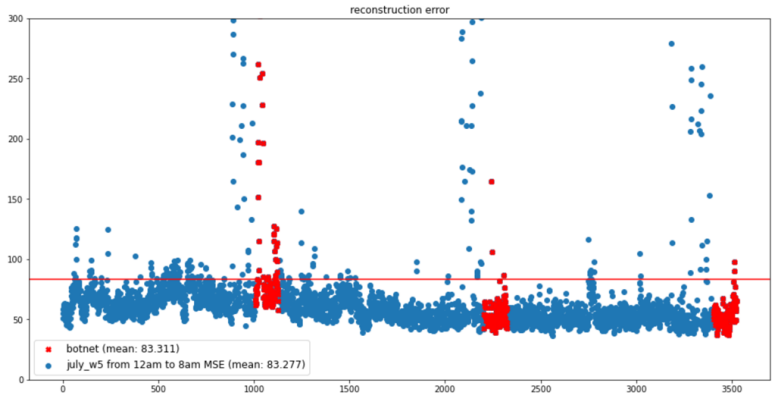


- Restrict the training on some specific time

  During different periods of time, the traffic may behave very differently. For example, the traffic during weekdays is usually with a larger size comparing to the weekend, traffic from work hours (e.g. 9AM - 5PM) has a larger volume than traffic from midnight or early morning (12 to 6AM). Observed that, the nerisbotnet is executed from 12-7AM during July week 5, therefore, we expect that training with data from 12 to 7AM each day might improve the performance in identifying nerisbotnet.

  I try training the model with restricted hours from June w1 weekdays, but the improvement in identifying nerisbotnet isn't significant. See the image below.

  - New experiment
  - Train with data from June w1, w2, weekdays, 12am-8am
  - Performance on identifying botnet improve to 32/183
  - # of red dots above red line / # of red dots = 55/366



Note that Due to the large size of the dataset, if we perform the restricted training, the size of the training tensors will be much less than the previous one, this could be affecting the result.

The limitation of this method is that it requires more computing resources than the 60 seconds tensor (to get all tensors for the same period of time, we need 6 times more time), therefore, it is very slow to get enough tensors for training.

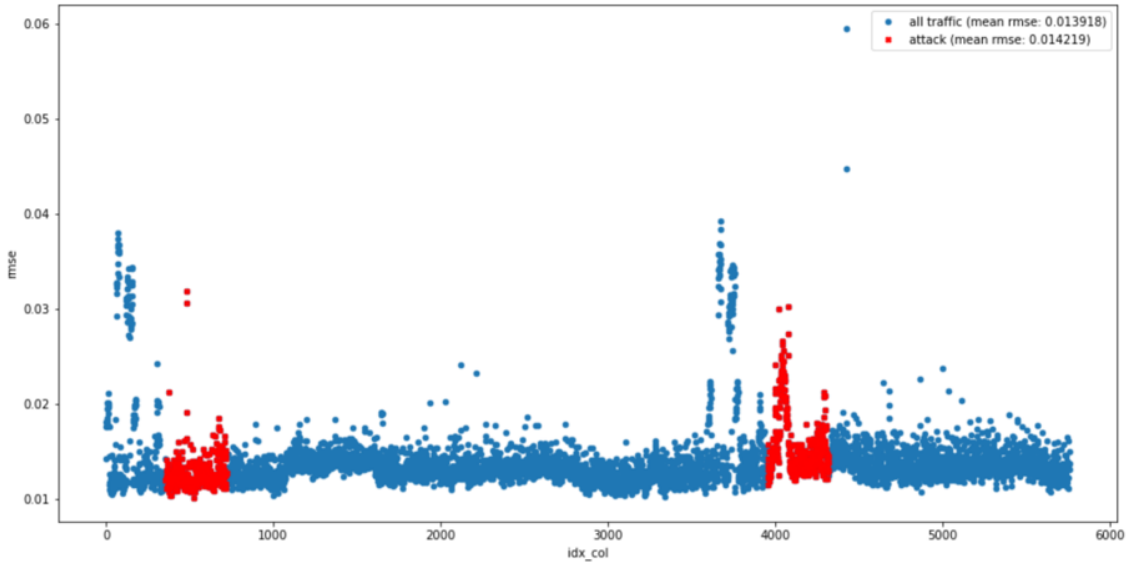
Comments on the model for identifying nerisbotnet: unlike the Dos attack, identifying the nerisbotnet is the most challenging part of the project, the experiments shown above can improve the performance in nerisbotnet identification, but the improvement is not significant, hence, we need to find new methods to make some non-trivial improvements.

3.3.3 Future work in improving the model

- Identifying the start and end of the (nerisbotnet) attack instead of identifying each individual attack tensor.

  Observation: the start and end of the nerisbotnet attack cause some bigger change in tensor reconstruction errors comparing to the attack tensor from another period of time. This can be observed from the following plot. The red points correspond to nerisbotnet tensors, one complete execution of the nerisbotnet attack is represented by a group of red points, we can see the start (or end) of the attack corresponds a spike in the plot, but in the middle of the attack the reconstruction error for the tensors go back to a normal level and it is, therefore, hard to detect based on one reconstruction error threshold.

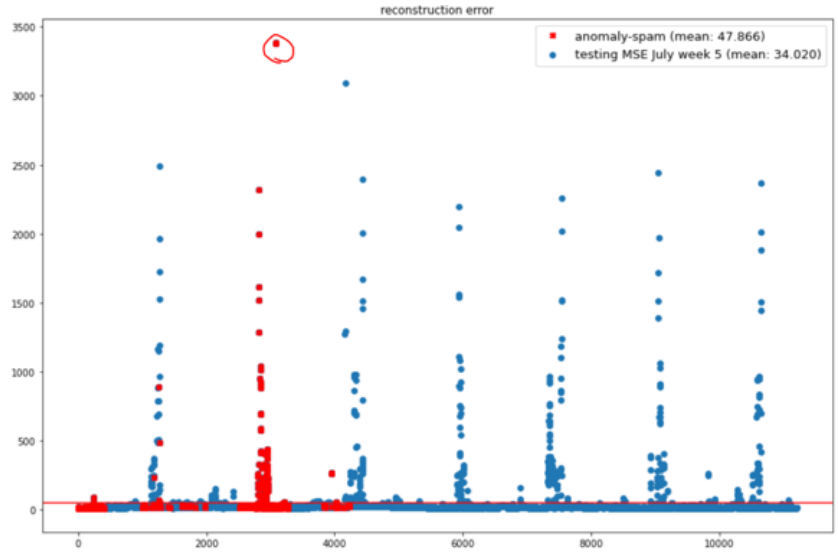
  This observation gives us a direction: we can identify the start and end of an attack and isolate the traffic within the interval for analysis, this could be useful in increasing the successful prediction rate.



- We are only considering part of information from the traffic, we are missing some of the important features such as the size of the traffic (i.e. `number_of_bytes` column from the table), we expect an improvement in performance if we change our tensors and include that information.

3.3.4 Traffic time series, Automated anomalous traffic isolation and Labeling

This section describes the other part of the IDS. The previous sections focus on building the system and evaluating it based on the labeled provided, however, after we explored the dataset more, we find some examples where the labeling provided by the UGR'16 is not accurate, for example, in the following tensor reconstruction error plot, the tensor in the red circle has starting timestamp: 2016-07-29 04:17:45 with 1-minute duration. It is labeled as anomaly-spam, and it has the largest reconstruction error among all tensors from July week 5.



The followings are parts of the traffic table for that tensor, we can see that only the first few rows in the table has anomaly-spam label, and it is not the cause of the high reconstruction error for the tensor.

timestamp	duration	src_IP	dest_IP	src_port	dest_port	protocol	flags	forwarding	type_of_se	packets_ex	number_of	label
7/29/2016 4:17	0.12	192.143.87.90	42.219.156.214	25	56975	TCP	.AP.SF	0	72	3	234	anomaly-spam
7/29/2016 4:17	0.124	192.143.87.120	42.219.156.214	25	51046	TCP	.AP.SF	0	72	3	234	anomaly-spam
7/29/2016 4:17	0.124	192.143.87.95	42.219.156.214	25	51241	TCP	.AP.SF	0	72	3	234	anomaly-spam
7/29/2016 4:17	0.136	108.66.255.255	42.219.156.214	25	58112	TCP	.AP.SF	0	72	3	234	anomaly-spam
7/29/2016 4:17	0.244	42.219.156.214	192.143.87.120	51046	25	TCP	.A.RS.	0	0	4	216	anomaly-spam
7/29/2016 4:17	0.244	42.219.156.214	192.143.87.90	56975	25	TCP	.A.RS.	0	0	4	216	anomaly-spam
7/29/2016 4:17	0.244	42.219.156.214	192.143.87.95	51241	25	TCP	.A.RS.	0	0	4	216	anomaly-spam
7/29/2016 4:17	0.26	42.219.156.214	108.66.255.255	58112	25	TCP	.A.RS.	0	0	4	216	anomaly-spam
7/29/2016 4:17	1.132	192.143.87.90	42.219.156.215	25	60210	TCP	.AP.SF	0	72	10	1129	anomaly-spam
7/29/2016 4:17	1.252	42.219.156.215	192.143.87.90	60210	25	TCP	.APRS.	0	0	10	1245	anomaly-spam
7/29/2016 4:17	1.056	108.66.255.255	42.219.156.215	25	57250	TCP	.AP.SF	0	72	10	1129	anomaly-spam
7/29/2016 4:17	1.176	42.219.156.215	108.66.255.255	57250	25	TCP	.APRS.	0	0	10	1245	anomaly-spam
7/29/2016 4:17	0.168	192.143.87.95	42.219.156.215	25	47297	TCP	.AP.SF	0	72	3	230	anomaly-spam
7/29/2016 4:17	0.292	42.219.156.215	192.143.87.95	47297	25	TCP	.A.RS.	0	0	4	216	anomaly-spam
7/29/2016 4:17	0.928	192.143.87.120	42.219.156.215	25	58083	TCP	.AP.SF	0	72	10	1129	anomaly-spam
7/29/2016 4:17	1.048	42.219.156.215	192.143.87.120	58083	25	TCP	.APRS.	0	0	10	1245	anomaly-spam
7/29/2016 4:18	0.124	108.66.255.255	42.219.156.214	25	38816	TCP	.AP.SF	0	72	3	234	anomaly-spam
7/29/2016 4:18	0.244	42.219.156.214	108.66.255.255	38816	25	TCP	.A.RS.	0	0	4	216	anomaly-spam
7/29/2016 4:18	0.124	192.143.87.90	42.219.156.214	25	52613	TCP	.AP.SF	0	72	3	234	anomaly-spam
7/29/2016 4:18	0.148	192.143.87.95	42.219.156.214	25	56216	TCP	.AP.SF	0	72	3	234	anomaly-spam
7/29/2016 4:18	0.164	192.143.87.120	42.219.156.214	25	47399	TCP	.AP.SF	0	72	3	234	anomaly-spam
7/29/2016 4:18	0.244	42.219.156.214	192.143.87.90	52613	25	TCP	.A.RS.	0	0	4	216	anomaly-spam
7/29/2016 4:18	0.268	42.219.156.214	192.143.87.95	56216	25	TCP	.A.RS.	0	0	4	216	anomaly-spam
7/29/2016 4:18	0.284	42.219.156.214	192.143.87.120	47399	25	TCP	.A.RS.	0	0	4	216	anomaly-spam
7/29/2016 4:18	0.988	108.66.255.199	42.219.156.215	25	45214	TCP	.AP.SF	0	72	10	1129	anomaly-spam
7/29/2016 4:18	1.108	42.219.156.215	108.66.255.199	45214	25	TCP	.APRS.	0	0	10	1245	anomaly-spam
7/29/2016 4:18	0.988	192.143.87.95	42.219.156.215	25	47004	TCP	.AP.SF	0	72	10	1129	anomaly-spam
7/29/2016 4:18	1.108	42.219.156.215	192.143.87.95	47004	25	TCP	.APRS.	0	0	10	1245	anomaly-spam
7/29/2016 4:18	0.172	192.143.87.90	42.219.156.215	25	56318	TCP	.AP.SF	0	72	3	230	anomaly-spam
7/29/2016 4:18	0.18	108.66.255.255	42.219.156.215	25	47076	TCP	.AP.SF	0	72	3	230	anomaly-spam
7/29/2016 4:18	0.292	42.219.156.215	192.143.87.90	56318	25	TCP	.A.RS.	0	0	4	216	anomaly-spam
7/29/2016 4:18	0.3	42.219.156.215	108.66.255.255	47076	25	TCP	.A.RS.	0	0	4	216	anomaly-spam
7/29/2016 4:17	5.92	42.219.156.211	168.42.141.224	80	60291	TCP	.AP.SF	0	0	6	1664	background
7/29/2016 4:17	6	42.219.159.197	216.32.204.62	55005	161	UDP	.A....	0	0	12	876	background
7/29/2016 4:17	60.028	176.206.157.1	42.219.153.27	2832	50578	UDP	.A....	0	0	22	2924	background
7/29/2016 4:17	6.104	42.219.156.211	176.244.134.122	80	40982	TCP	.AP.SF	0	0	5	1612	background

The cause of the high reconstruction corresponds to the following table, it shows a malicious pattern but in the original file, it is only labeled as background. The traffic behavior is one machine (62.100.57.109) is

sending one packet to different ports from the same destination IP.

7/29/2016 4:18	0 62.100.57.109	42.219.152.114	5066	5063 UDP	.A....	0	0	1	441 background
7/29/2016 4:18	0 62.100.57.109	42.219.152.114	5066	5064 UDP	.A....	0	0	1	441 background
7/29/2016 4:18	0 62.100.57.109	42.219.152.114	5066	5065 UDP	.A....	0	0	1	442 background
7/29/2016 4:18	0 62.100.57.109	42.219.152.114	5066	5066 UDP	.A....	0	0	1	442 background
7/29/2016 4:18	0 62.100.57.109	42.219.152.114	5066	5067 UDP	.A....	0	0	1	437 background
7/29/2016 4:18	0 62.100.57.109	42.219.152.114	5066	5068 UDP	.A....	0	0	1	441 background
7/29/2016 4:18	0 62.100.57.109	42.219.152.114	5066	5069 UDP	.A....	0	0	1	437 background
7/29/2016 4:18	0 62.100.57.109	42.219.152.114	5066	5070 UDP	.A....	0	0	1	438 background
7/29/2016 4:18	0 62.100.57.109	42.219.152.114	5066	5071 UDP	.A....	0	0	1	440 background
7/29/2016 4:18	0 62.100.57.109	42.219.152.114	5066	5072 UDP	.A....	0	0	1	439 background
7/29/2016 4:18	0 62.100.57.109	42.219.152.114	5066	5073 UDP	.A....	0	0	1	442 background
7/29/2016 4:18	0 62.100.57.109	42.219.152.114	5066	5074 UDP	.A....	0	0	1	441 background
7/29/2016 4:18	0 62.100.57.109	42.219.152.114	5066	5075 UDP	.A....	0	0	1	441 background
7/29/2016 4:18	0 62.100.57.109	42.219.152.114	5066	5076 UDP	.A....	0	0	1	440 background
7/29/2016 4:18	0 62.100.57.109	42.219.152.114	5066	5077 UDP	.A....	0	0	1	442 background
7/29/2016 4:18	0 62.100.57.109	42.219.152.114	5066	5078 UDP	.A....	0	0	1	440 background
7/29/2016 4:18	0 62.100.57.109	42.219.152.114	5066	5079 UDP	.A....	0	0	1	442 background
7/29/2016 4:18	0 62.100.57.109	42.219.152.114	5066	5080 UDP	.A....	0	0	1	438 background
7/29/2016 4:18	0 62.100.57.109	42.219.152.114	5066	5081 UDP	.A....	0	0	1	441 background
7/29/2016 4:18	0 62.100.57.109	42.219.152.114	5066	5082 UDP	.A....	0	0	1	441 background
7/29/2016 4:18	0 62.100.57.109	42.219.152.114	5066	5083 UDP	.A....	0	0	1	441 background
7/29/2016 4:18	0 62.100.57.109	42.219.152.114	5066	5084 UDP	.A....	0	0	1	440 background
7/29/2016 4:18	0 62.100.57.109	42.219.152.114	5066	5085 UDP	.A....	0	0	1	441 background
7/29/2016 4:18	0 62.100.57.109	42.219.152.11	5066	5061 UDP	.A....	0	0	1	439 background
7/29/2016 4:18	0 62.100.57.109	42.219.152.11	5066	5062 UDP	.A....	0	0	1	442 background
7/29/2016 4:18	0 62.100.57.109	42.219.152.11	5066	5063 UDP	.A....	0	0	1	441 background
7/29/2016 4:18	0 62.100.57.109	42.219.152.11	5066	5064 UDP	.A....	0	0	1	440 background
7/29/2016 4:18	0 62.100.57.109	42.219.152.11	5066	5065 UDP	.A....	0	0	1	440 background
7/29/2016 4:18	0 62.100.57.109	42.219.152.11	5066	5066 UDP	.A....	0	0	1	438 background
7/29/2016 4:18	0 62.100.57.109	42.219.152.11	5066	5067 UDP	.A....	0	0	1	442 background
7/29/2016 4:18	0 62.100.57.109	42.219.152.11	5066	5068 UDP	.A....	0	0	1	441 background
7/29/2016 4:18	0 62.100.57.109	42.219.152.11	5066	5069 UDP	.A....	0	0	1	441 background
7/29/2016 4:18	0 62.100.57.109	42.219.152.11	5066	5070 UDP	.A....	0	0	1	441 background

Therefore, we think that if we can use our system to find some of the traffic records similar to the above example and label them, it can contribute to the UGR'16 dataset.

In order to find the root cause of the anomaly behavior of the traffic, or what is causing the high reconstruction error for the tensors, we introduce the following concept.

Our tensor has a shape of $64 \times 64 \times 64$, and we can view the tensor as the combination of 64^3 time series, for example, the time series (42.219, 105.165, UDP/TCP:53) has different count values for different timestamps. The reconstruction error for a tensor is the average reconstruction error for all 64^3 time series, by focusing on the anomaly time series which have the large reconstruction error (greater than some threshold), we can identify which parts of the traffic behavior differently than normal traffic and hence we can find the root cause for the anomaly. The following shows the top 15 anomaly time series that has the largest reconstruction error:

```
tensor # 3075 ; timestamp: 2016-07-29 04:17:45
reconstruction difference: 10751 ; idx: ((56, 63), '42.219.147', (4096, 5119))
reconstruction difference: 10460 ; idx: ((56, 63), '42.219.148', (4096, 5119))
reconstruction difference: 10433 ; idx: ((56, 63), '42.219.149', (4096, 5119))
reconstruction difference: 8598 ; idx: ((56, 63), '42.219.146', (4096, 5119))
reconstruction difference: 7704 ; idx: ((56, 63), '42.219.151', (4096, 5119))
reconstruction difference: 6147 ; idx: ((56, 63), '42.219.159', (4096, 5119))
reconstruction difference: 6144 ; idx: ((56, 63), '42.219.157', (4096, 5119))
reconstruction difference: 6084 ; idx: ((56, 63), '42.219.153', (4096, 5119))
reconstruction difference: 6024 ; idx: ((56, 63), '42.219.158', (4096, 5119))
reconstruction difference: 5968 ; idx: ((56, 63), '42.219.150', (4096, 5119))
reconstruction difference: 2036 ; idx: ((64, 71), '42.219.154', (4096, 5119))
reconstruction difference: 913 ; idx: ((64, 71), '42.219.153', (4096, 5119))
reconstruction difference: 464 ; idx: ((64, 71), '42.219.155', (443, ('TCP', 'UDP')))
reconstruction difference: 448 ; idx: ((56, 63), '42.219.147', ((5060, 5061), ('TCP', 'UDP')))
reconstruction difference: 447 ; idx: ((32, 39), '42.219.149', (1024, 2047))
```

Based on the list of the anomaly time series, we can implement an Automated Suspicious (Attack) Traffic Identification Algorithm:

- 1) Choose the next most anomalous time-series from the list.
- 2) Isolate traffic that matches the time-series.
- 3) Calculate entropy for each attribute (e.g. src_IP) in the isolated traffic.
  - Choose the one with the lowest entropy
  - If there is a value with a recurring rate of more than X%, choose the value1 with the highest recurring rate, isolate associated traffic. (go to step 1 if there is no such value)
  - Calculate entropy for the remaining attributes in the isolated traffic. Choose the one with the lowest entropy If there is a value with a recurring rate of more than Y%, choose the value2 with the highest recurring rate, isolate associated traffic
- 4) Remove the traffic that matches value1 and value2 and go to step 3 if the tensor is still anomalous.

The following table shows some anomaly traffic isolated by the algorithm

timestamp	duration	src_IP	dest_IP	src_port	dest_port	protocol	flags	forwarding	type_of_service	packets_exchanged	number_of_bytes	label
1469679709	0	42.219.150.246	42.219.158.16	13547	80	TCP	...RS.	0	0	2	200	dos
1469679709	0	42.219.150.246	42.219.158.16	13550	80	TCP	...RS.	0	0	2	200	dos
1469679709	0	42.219.150.246	42.219.158.16	13552	80	TCP	...RS.	0	0	2	200	dos
1469679709	0	42.219.150.246	42.219.158.16	13553	80	TCP	...RS.	0	0	2	200	dos
1469679709	0	42.219.150.246	42.219.158.16	13554	80	TCP	...RS.	0	0	2	200	dos
1469679709	0	42.219.150.246	42.219.158.16	13555	80	TCP	...RS.	0	0	2	200	dos
1469679709	0	42.219.150.246	42.219.158.16	13556	80	TCP	...RS.	0	0	2	200	dos
1469679709	0	42.219.150.246	42.219.158.16	13557	80	TCP	...RS.	0	0	2	200	dos
1469679709	0	42.219.150.246	42.219.158.16	13558	80	TCP	...RS.	0	0	2	200	dos
1469679709	0	42.219.150.246	42.219.158.16	13559	80	TCP	...RS.	0	0	2	200	dos
1469679709	0	42.219.150.246	42.219.158.16	13560	80	TCP	...RS.	0	0	2	200	dos
1469679709	0	42.219.150.246	42.219.158.16	13561	80	TCP	...RS.	0	0	2	200	dos
1469679709	0	42.219.150.246	42.219.158.16	13562	80	TCP	...RS.	0	0	2	200	dos
1469679709	0	42.219.150.246	42.219.158.16	13563	80	TCP	...RS.	0	0	2	200	dos
1469679709	0	42.219.150.246	42.219.158.16	13564	80	TCP	...RS.	0	0	2	200	dos
1469679709	0	42.219.150.246	42.219.158.16	13565	80	TCP	...RS.	0	0	2	200	dos
1469679709	0	42.219.150.246	42.219.158.16	13566	80	TCP	...RS.	0	0	2	200	dos
1469679709	0	42.219.150.246	42.219.158.16	13567	80	TCP	...RS.	0	0	2	200	dos
1469679709	0	42.219.150.246	42.219.158.16	13568	80	TCP	...RS.	0	0	2	200	dos
1469679709	0	42.219.150.246	42.219.158.16	13570	80	TCP	...RS.	0	0	2	200	dos
1469679709	0	42.219.150.246	42.219.158.16	13571	80	TCP	...RS.	0	0	2	200	dos
1469679709	0	42.219.150.246	42.219.158.16	13572	80	TCP	...RS.	0	0	2	200	dos
1469679709	0	42.219.150.246	42.219.158.16	13573	80	TCP	...RS.	0	0	2	200	dos
1469679709	0	42.219.150.246	42.219.158.16	13574	80	TCP	...RS.	0	0	2	200	dos
1469679709	0	42.219.150.246	42.219.158.16	13575	80	TCP	...RS.	0	0	2	200	dos
1469679709	0	42.219.150.246	42.219.158.16	13577	80	TCP	...RS.	0	0	2	200	dos
1469679709	0	42.219.150.246	42.219.158.16	13578	80	TCP	...RS.	0	0	2	200	dos
1469679709	0	42.219.150.246	42.219.158.16	13579	80	TCP	...RS.	0	0	2	200	dos
1469679709	0	42.219.150.246	42.219.158.16	13580	80	TCP	...RS.	0	0	2	200	dos
1469679709	0	42.219.150.246	42.219.158.16	13581	80	TCP	...RS.	0	0	2	200	dos
1469679709	0	42.219.150.246	42.219.158.16	13582	80	TCP	...RS.	0	0	2	200	dos
1469679709	0	42.219.150.246	42.219.158.16	13583	80	TCP	...RS.	0	0	2	200	dos
1469679709	0	42.219.150.246	42.219.158.16	13585	80	TCP	...RS.	0	0	2	200	dos
1469679709	0	42.219.150.246	42.219.158.16	13586	80	TCP	...RS.	0	0	2	200	dos
1469679709	0	42.219.150.246	42.219.158.16	13587	80	TCP	...RS.	0	0	2	200	dos
1469679709	0	42.219.150.246	42.219.158.16	13589	80	TCP	...RS.	0	0	2	200	dos
1469679709	0	42.219.150.246	42.219.158.16	13591	80	TCP	...RS.	0	0	2	200	dos
1469679709	0	42.219.150.246	42.219.158.16	13592	80	TCP	...RS.	0	0	2	200	dos
1469679709	0	42.219.150.246	42.219.158.16	13593	80	TCP	...RS.	0	0	2	200	dos
1469679709	0	42.219.150.246	42.219.158.16	13594	80	TCP	...RS.	0	0	2	200	dos

The following is the report generated by the algorithm along with the above table.

```
1469679709
july_w5/tensor1469679709.pt ; timestamp: 2016-07-28 00:21:49
Original overall reconstruction error: 0.028034735471010208

size of the 1-min background traffic: 108032
Time series reconstruction err: 11.116220474243164 ; idx: ('42.219.150', '42.219.158', (80, 'TCP'))
Entropy of src_IP: 0.6931471805599453
Entropy of dest_IP: 0.0
Entropy of dest_port: 0.0
value1: 42.219.158.16 ; value2: 80
new overall reconstruction error: 0.019954001531004906
original 1-min df modified!
Finish analyzing time series:  ('42.219.150', '42.219.158', (80, 'TCP'))

size of the 1-min background traffic: 95944
Time series reconstruction err: 5.978370666503906 ; idx: ('42.219.158', '42.219.150', (16384, 22527))
Entropy of src_IP: 0.0
Entropy of dest_IP: 0.6930534668238298
Entropy of dest_port: 8.096053434220718
value1: 42.219.158.16 ; value2: 42.219.150.247
new overall reconstruction error: 0.016934458166360855
original 1-min df modified!
value1: 42.219.150.247 ; value2: 18431
new overall reconstruction error: 0.01403056364506483
Removing value1: 42.219.150.247 and value2: 18431 resulting in decreasing tensor reconstruction error, but do not less than the threshold.
Finish analyzing time series:  ('42.219.158', '42.219.150', (16384, 22527))

size of the 1-min background traffic: 92649
Time series reconstruction err: 5.136929512023926 ; idx: ('42.219.158', '42.219.150', (10240, 16383))
Entropy of src_IP: 0.0
Entropy of dest_IP: 0.6930173813791283
Entropy of dest_port: 7.946039350809194
value1: 42.219.158.16 ; value2: 42.219.150.246
new overall reconstruction error: 0.013961697928607464
original 1-min df modified!
value1: 42.219.150.246 ; value2: 14333
new overall reconstruction error: 0.012572051025927067
Removing value1: 42.219.150.246 and value2: 14333 resulting in decreasing tensor reconstruction error, and is less than the original threshold.
By removing a sequence of value 1 & 2 pair, the reconstruction error of the tensor reduce below the threshold!
Finish analyzing time series:  ('42.219.158', '42.219.150', (10240, 16383))

size of the 1-min background traffic: 89811
Time series reconstruction err: 1.786798119544983 ; idx: ('42.219.154', '143.72', (53, ('TCP', 'UDP')))
By removing a sequence of value 1 & 2 pair, the reconstruction error of the tensor reduce below the threshold!
Finish analyzing time series:  ('42.219.154', '143.72', (53, ('TCP', 'UDP')))

size of the 1-min background traffic: 89811
Time series reconstruction err: 1.4070000648498535 ; idx: ('42.219.158', '143.72', (53, ('TCP', 'UDP')))
By removing a sequence of value 1 & 2 pair, the reconstruction error of the tensor reduce below the threshold!
Finish analyzing time series:  ('42.219.158', '143.72', (53, ('TCP', 'UDP')))
```

Here is another example output from the algorithm, it shows part of a scan traffic on port 1433, but they are labeled as background. The traffic behavior is one machine (247.191.222.230) might be doing some vulnerability scan on port 1443 of different destination IP, each flow record has a very short duration and there is only one packet exchanged and only 48 bytes are transferred.

timestamp	duration	src_IP	dest_IP	src_port	dest_port	protocol	flags	forwarding_status	type_of_service	packets_exchanged	number_of_bytes	label
1469679254	0	247.191.222.230	42.219.151.248	35853	1433	TCP	....S.	0	0	1	48	background
1469679255	0	247.191.222.230	42.219.151.102	35858	1433	TCP	....S.	0	4	1	48	background
1469679255	0	247.191.222.230	42.219.151.116	35857	1433	TCP	....S.	0	0	1	48	background
1469679255	0	247.191.222.230	42.219.151.121	35856	1433	TCP	....S.	0	0	1	48	background
1469679255	0	247.191.222.230	42.219.151.1	36123	1433	TCP	....S.	0	4	1	48	background
1469679255	0	247.191.222.230	42.219.151.226	35855	1433	TCP	....S.	0	4	1	48	background
1469679255	0	247.191.222.230	42.219.151.236	35854	1433	TCP	....S.	0	4	1	48	background
1469679255	0	247.191.222.230	42.219.151.248	35853	1433	TCP	....S.	0	0	1	48	background
1469679255	0	247.191.222.230	42.219.151.66	35862	1433	TCP	....S.	0	0	1	48	background
1469679255	0	247.191.222.230	42.219.151.68	35860	1433	TCP	....S.	0	0	1	48	background
1469679255	0	247.191.222.230	42.219.151.70	35861	1433	TCP	....S.	0	0	1	48	background
1469679255	0	247.191.222.230	42.219.151.85	35859	1433	TCP	....S.	0	4	1	48	background
1469679256	0	247.191.222.230	42.219.151.102	35858	1433	TCP	....S.	0	4	1	48	background
1469679256	0	247.191.222.230	42.219.151.1	36123	1433	TCP	....S.	0	4	1	48	background
1469679256	0	247.191.222.230	42.219.151.66	35862	1433	TCP	....S.	0	0	1	48	background
1469679256	0	247.191.222.230	42.219.151.68	35860	1433	TCP	....S.	0	0	1	48	background
1469679256	0	247.191.222.230	42.219.151.70	35861	1433	TCP	....S.	0	0	1	48	background
1469679256	0	247.191.222.230	42.219.151.85	35859	1433	TCP	....S.	0	4	1	48	background
1469679257	0	247.191.222.230	42.219.151.116	40133	1433	TCP	....S.	0	0	1	48	background
1469679257	0	247.191.222.230	42.219.151.121	40132	1433	TCP	....S.	0	0	1	48	background
1469679257	0	247.191.222.230	42.219.151.152	37215	1433	TCP	....S.	0	0	1	48	background
1469679257	0	247.191.222.230	42.219.151.166	39571	1433	TCP	....S.	0	0	1	48	background
1469679257	0	247.191.222.230	42.219.151.175	37226	1433	TCP	....S.	0	0	1	48	background
1469679257	0	247.191.222.230	42.219.151.182	38834	1433	TCP	....S.	0	4	1	48	background
1469679257	0	247.191.222.230	42.219.151.226	40135	1433	TCP	....S.	0	4	1	48	background
1469679257	0	247.191.222.230	42.219.151.236	40124	1433	TCP	....S.	0	4	1	48	background
1469679257	0	247.191.222.230	42.219.151.248	40125	1433	TCP	....S.	0	0	1	48	background
1469679257	2.832	247.191.222.230	42.219.151.127	37736	1433	TCP	....S.	0	4	2	96	background
1469679257	2.832	247.191.222.230	42.219.151.232	39656	1433	TCP	....S.	0	0	2	96	background
1469679257	2.856	247.191.222.230	42.219.151.117	36963	1433	TCP	....S.	0	4	2	96	background
1469679257	2.856	247.191.222.230	42.219.151.125	39665	1433	TCP	....S.	0	0	2	96	background

The main idea of the algorithm is by focusing on each anomalous time series from the list and finding the value1-value2 pair based on entropy that is the most likely cause of the high reconstruction error of the tensor. So that we can narrow down the 1-minute traffic and find the anomalous traffic mixed in it. According to

the output generated, the algorithm has a good performance in identifying some obvious attacks like Dos. It is also able to identify some unlabeled attack. By setting the parameter, we can decide the "strictness" of the algorithm, i.e. the degree of anomaly that is qualified as anomalous traffic, the higher the "strictness" we set, the lower the false positive isolation we will make.

Finally, the anomaly time series reconstruction error can also be used to define the threshold for distinguishing between normal and abnormal tensors. For example, instead of using only one overall reconstruction error threshold for distinguishing normal and abnormal tensors, we can take into account the anomaly time series reconstruction error and implement some threshold-based rules for identifying anomaly.

3.3.5 Training and labeling in a recursive manner

Once we finish building our system, the plan for applying our model is to traverse through the whole UGR'16 dataset by the following method:

Train with data from the first two weeks, then apply the model to the following week, isolating any potential anomalous traffic, and train with week #2 and week #3, evaluating on week #4.

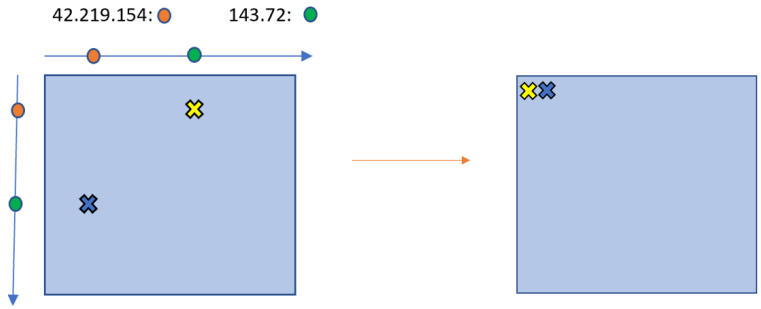
Recursively apply the process to let the model adapt to the varying behavior for each adjacent weeks so that the prediction performance can be improved since the model is less distracted by the different behavior of the traffic from different weeks.

During the process, the model will generate multiple suspicious traffic files that are unlabeled or labeled as background in the original UGR'16 dataset, as I mentioned above, this could be a contribution to the UGR'16 dataset.

3.4 New Tensor Specification

In this section, I will describe a new tensor format that might improve the performance of the IDS.

The key change is we rearrange the positions of each time series (cells in the tensor) in a way that the time series that are more likely to be correlated to each other are put into some new spots that are close to each other. For instance, each row in the original UGR'16 table corresponds to a uni-directional flow, e.g. from A (source IP) to B (destination IP). For all the flows from B to A, we should consider they are more likely to be correlated to all the flows from A to B comparing to any other flows. Therefore, in the new tensor, by putting them close to each other such that the kernel of the CNN layer can capture them, we could take the advantage of CNN's ability to capture the local property of data. And the model might learn some pattern related to the correlation between different time series (e.g. bi-directional flow).



References

[1] C. Zhang, D. Song, Y. Chen, X. Feng, C. Lumezanu, W. Cheng, J. Ni, B. Zong, H. Chen, and N. V. Chawla, "A Deep Neural Network for Unsupervised Anomaly Detection and Diagnosis in Multivariate Time Series Data," Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, pp. 1409–1416, 2019.

[2] R. Holt, S. Aubrey, A. DeVille, W. Haight, T. Gary and Q. Wang. "Deep Autoencoder Neural Networks for Detecting Lateral Movement in Computer Networks," Proceedings on the International Conference on Artificial Intelligence (ICAI); Athens, 2019.

[3] G. Maciá-Fernández, J. Camacho, R. Magán-Carrión, P. García-Teodoro and R. Theron, "UGR'16: a new dataset for the evaluation of cyclostationarity-based network IDSs", In Computers & Security, 2017.