Topic: Convolutional Neural Network Based Intrusion Detection System

CSE534-Project-Progress Report, Fall 2019

Instructor: Aruna Balasubramanian

TA: Javad Nejati

Student: Jiaxiang Ren (112748304)

Email: jiaxiang.ren@stonybrook.edu

Repo link: https://github.com/reckdk/Networks_Project

Introduction, Problem Statement, Approach

Background

Intrusion detection is a crucial task in information security. With the increment of network traffic and development of attacks against detection, it becomes more and more difficult to categorize the normal and malicious network traffic. Recently, some researchers take advantage of deep learning by incorporating Convolutional Neural Network (CNN) into Intrusion Detection (ID) and achieves good accuracy.

Problem Statement

ID is to identify network invasions that could be an ongoing invasion, or even an intrusion that has already occurred. Indeed, intrusion detection can be taken as a classification task, such as a binary (normal/attack) or a multi-class (normal and different kinds of attack) classification task. Naturally, machine learning approaches have been widely used in ID and achieved relative high performances.

Approach

However, there is a crucial issue for the performance of classic machine learning (ML) methods, i.e., the good features requiring considerable expertise. It usually takes lots of time and resources to design a kind of robust feature. Deep learning and Convolutional Neural Networks (CNNs) [1] provide a solution to automatically extract high-level features from low-level ones and gain powerful representation for inference.

Due to the great success of deep learning in the field of computer vision, some research using deep learning approach for ID have recently emerged and some of them have outperformed classic machine learning methods. One main reason is that CNNs are capable of extracting effective spatial features from hierarchical structures. Specifically, network traffic has an obvious hierarchical structure: TCP connection, flow, session, service and host [2]. As a result, many researchers use CNNs in intrusion detection systems. Cui et al. [3] made a systematic comparison of CNN and RNN on the deep learning based intrusion detection systems, aiming to guide for DNN selection. CNN model mainly contains two modules: the basic convolutional module as feature extractor and the fully connected header as classifier. However, one of the most important module of CNNs - i.e., the header (classifier) and how to design it are not discussed. Furthermore, some state-of-the-art techniques have not been tested yet and the classification accuracy might be improved further. Besides, some data in ISCX IDS 2012 dataset show artifacts which may influence the robustness of detection system. In this project, I study on these three points to explore how to improve the performance and robustness of CNNs for ID.

Solution (what exactly did you do)

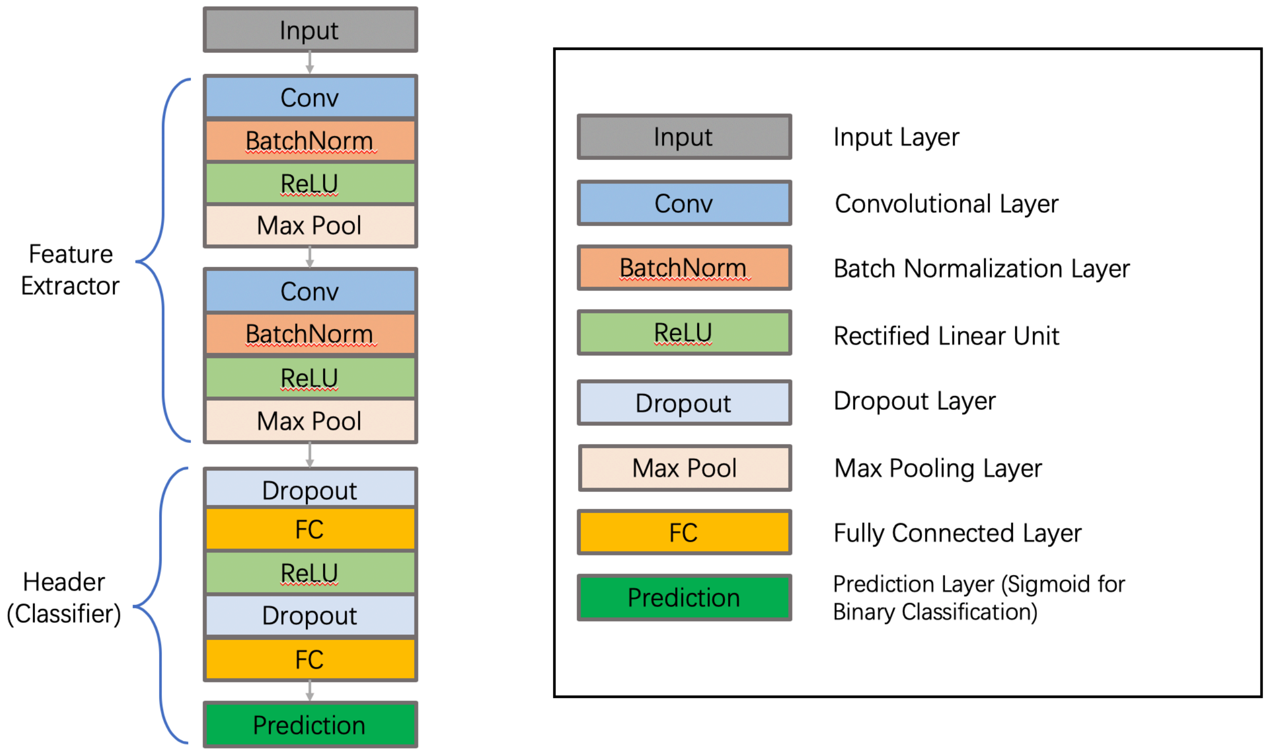
As mentioned above, this project is firstly going to check whether some generally admitted techniques in deep learning could boost the performance and robustness for intrusion detection. This part contain experiments on the two modules of CNNs (feature extractor and classifier).

The second part involves the influence of different data preprocessing and augmentation methods for intrusion detection systems.

CNNs

Simple Introduction for CNN

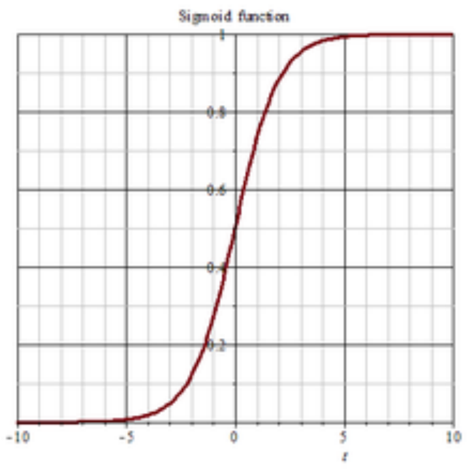
This subsection introduces CNN simply for understanding what I have modified on previous works. The figure below is an example of CNN.



Example of CNN Frameworks.

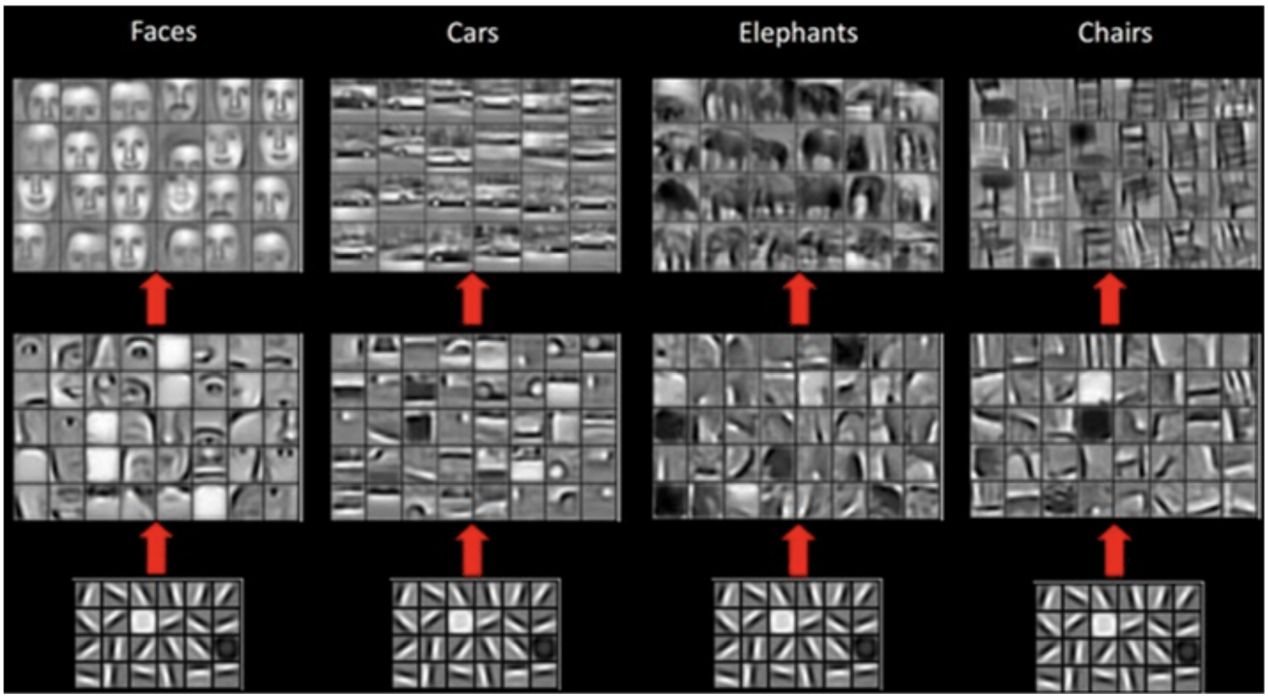
There mainly 7 different types of layers in this example. Each of these layers has different parameters that can be optimized and performs a different task on the input data.

- Convolutional Layer: Convolutional layers are the layers where filters are applied to the input image in the first layer or to other feature maps in layers before it. The most important parameters here are the number of kernels and the size of the kernels.
- Batch Normalization Layer: Feature normalization layers which could boost the training of model.
- Rectified Linear Unit Layer: This is an activate layer with fast gradient computation and backwards.
- Max Pooling Layer: Pooling layers are similar to convolutional layers, but they perform the max pooling operation, which takes the maximum value in a certain filter region as output. These are typically used to reduce the dimensionality of the network.
- Dropout Layer: Dropout Layer block features with probability p (parameters needing finetuning). This layer acts like ensemble learning.
- Fully Connected Layer: Fully connected layers are placed after the feature extractor of CNN and act like weights of classifier.
- Prediction Layer: For binary classification task, this layer is a Sigmoid function (See figure below). For multi-class classification task, this layer acts like softmax function.



Sigmoid Function. (Figure from Wiki)

CNNs usually learn basic texture features in lower layers and semantic features in layers near prediction layer.



Examples of CNN’s trained to recognize specific objects and their generated feature maps. (Figure from https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac)

Model Design

I take a popular CNN model - VGG16 from the vision group of Oxford University as basic model. In original VGG16 model, there are three FC layers in classifier with 4096, 4096 and 1000 neurons respectively. Such heavy header is at high risk of overfitting so I remove one FC layers and add one Dropout layer to reduce the risk of overfitting. The modified version is called Prototype Model, which reduces the classifier size into 1/175:

$$\begin{gathered}512 * 4096 + 4096 * 4096 + 4096 * 1000 = 22970368 \\ 512 * 256 + 256 * 1 = 131328 \\ 22970368/131328. \approx 175\end{gathered} \tag{1}$$

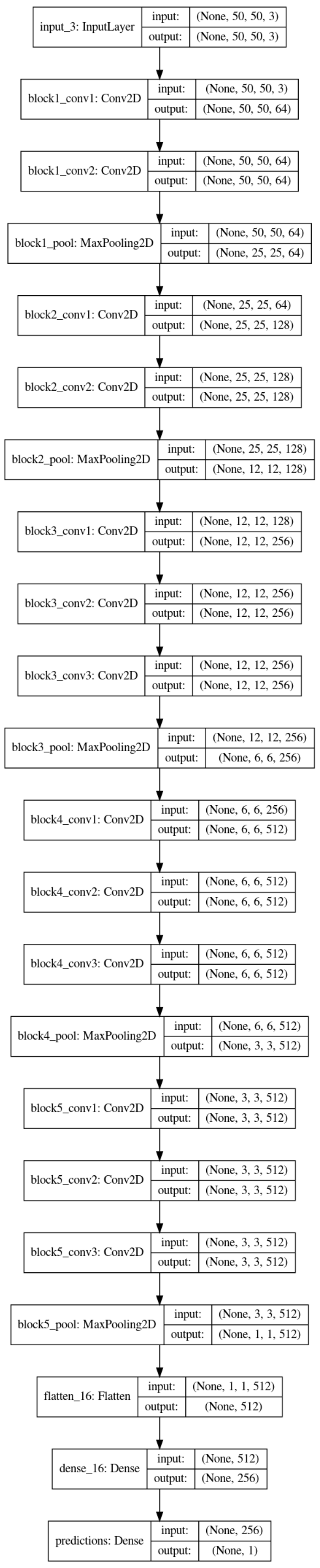
1. Prototype Model

The parameters of each layer in the modified VGG16 model are shown as below. For simplicity, the activation layers are not shown. Please note that 'Dense' layers here mean 'Fully Connected Layers' (Keras notation).

The input size is $50 \times 50 \times 3$ (the first 7500 bytes of destination payload) which is labeled as normal or malicious (Details are in Experiment Settings Section). The output size is 1, which means the malicious probability of an input sample.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_3 (InputLayer)         (None, 50, 50, 3)         0         
```

```
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 50, 50, 64)        1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 50, 50, 64)        36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 25, 25, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 25, 25, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 25, 25, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 12, 12, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 12, 12, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 12, 12, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 12, 12, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 6, 6, 256)         0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 6, 6, 512)         1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 6, 6, 512)         2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 6, 6, 512)         2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 3, 3, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 3, 3, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 3, 3, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 3, 3, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 1, 1, 512)         0
_________________________________________________________________
flatten_16 (Flatten)         (None, 512)               0
_________________________________________________________________
dense_16 (Dense)             (None, 256)               131328
_________________________________________________________________
predictions (Dense)          (None, 1)                 257
=================================================================
Total params: 14,846,273
Trainable params: 131,585
Non-trainable params: 14,714,688
_________________________________________________________________
```


input_3: InputLayer
input: (None, 50, 50, 3)
output: (None, 50, 50, 3)
block1_conv1: Conv2D
input: (None, 50, 50, 3)
output: (None, 50, 50, 64)
block1_conv2: Conv2D
input: (None, 50, 50, 64)
output: (None, 50, 50, 64)
block1_pool: MaxPooling2D
input: (None, 50, 50, 64)
output: (None, 25, 25, 64)
block2_conv1: Conv2D
input: (None, 25, 25, 64)
output: (None, 25, 25, 128)
block2_conv2: Conv2D
input: (None, 25, 25, 128)
output: (None, 25, 25, 128)
block2_pool: MaxPooling2D
input: (None, 25, 25, 128)
output: (None, 12, 12, 128)
block3_conv1: Conv2D
input: (None, 12, 12, 128)
output: (None, 12, 12, 256)
block3_conv2: Conv2D
input: (None, 12, 12, 256)
output: (None, 12, 12, 256)
block3_conv3: Conv2D
input: (None, 12, 12, 256)
output: (None, 12, 12, 256)
block3_pool: MaxPooling2D
input: (None, 12, 12, 256)
output: (None, 6, 6, 256)
block4_conv1: Conv2D
input: (None, 6, 6, 256)
output: (None, 6, 6, 512)
block4_conv2: Conv2D
input: (None, 6, 6, 512)
output: (None, 6, 6, 512)
block4_conv3: Conv2D
input: (None, 6, 6, 512)
output: (None, 6, 6, 512)
block4_pool: MaxPooling2D
input: (None, 6, 6, 512)
output: (None, 3, 3, 512)
block5_conv1: Conv2D
input: (None, 3, 3, 512)
output: (None, 3, 3, 512)
block5_conv2: Conv2D
input: (None, 3, 3, 512)
output: (None, 3, 3, 512)
block5_conv3: Conv2D
input: (None, 3, 3, 512)
output: (None, 3, 3, 512)
block5_pool: MaxPooling2D
input: (None, 3, 3, 512)
output: (None, 1, 1, 512)
flatten_16: Flatten
input: (None, 1, 1, 512)
output: (None, 512)
dense_16: Dense
input: (None, 512)
output: (None, 256)
predictions: Dense
input: (None, 256)
output: (None, 1)

VGG16 Based Prototype Model Framework.

Though reducing the size of classifier, VGG16 is still a relative large CNN model and there are 131,585 trainable parameters in total.

2. Model with Modified Header

The header design in Prototype Model is quite naive - i.e., reduce size as many as possible. Some powerful and widely used modules, such as Dropout, BatchNormalization and Parametric Rectified Linear Unit, are not included. In the modified header, these modules are introduced to reduce the risk of overfitting and improve performance. The modified header is as below.

```
... ...
flatten_3 (Flatten)          (None, 512)               0
_________________________________________________________________
batch_normalization_1 (Batch (None, 512)               2048
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_5 (Dense)              (None, 256)               131328
_________________________________________________________________
p_re_lu_1 (PReLU)            (None, 256)               256
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
batch_normalization_2 (Batch (None, 256)               1024
_________________________________________________________________
predictions (Dense)          (None, 1)                 257
=================================================================
Total params: 14,849,601
Trainable params: 133,377
Non-trainable params: 14,716,224
_________________________________________________________________
```

The total trainable parameters are 133,377 which are slightly more than Prototype Model. The additional parameters are introduced by Dropout, BatchNormalization and Parametric Rectified Linear Unit.

3. Light Weight Model

From previous setting, we can see that such models are quite large and have tremendous trainable parameters, which is likely to overfit on specific dataset. So I design a light weight model specialized for this task. The model structure is as below.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 32, 32, 3)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 12)        912
_________________________________________________________________
batch_normalization_1 (Batch (None, 32, 32, 12)        48
_________________________________________________________________
activation_1 (Activation)    (None, 32, 32, 12)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 32, 12)        1308
```

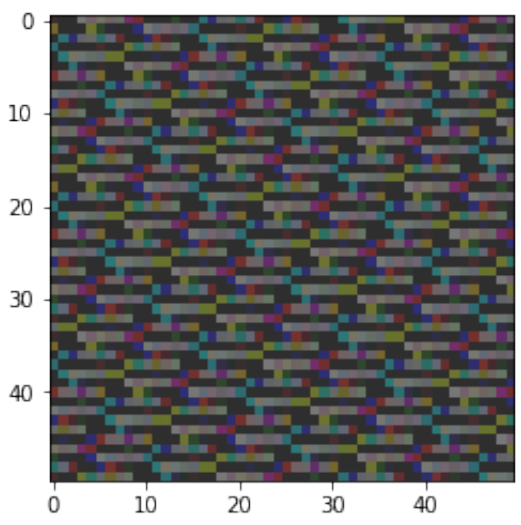
```
_________________________________________________________________
batch_normalization_2 (Batch (None, 32, 32, 12)        48        
_________________________________________________________________
activation_2 (Activation)    (None, 32, 32, 12)        0         
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 12)        0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 24)        2616      
_________________________________________________________________
batch_normalization_3 (Batch (None, 16, 16, 24)        96        
_________________________________________________________________
activation_3 (Activation)    (None, 16, 16, 24)        0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 8, 8, 48)          10416     
_________________________________________________________________
batch_normalization_4 (Batch (None, 8, 8, 48)          192       
_________________________________________________________________
activation_4 (Activation)    (None, 8, 8, 48)          0         
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 8, 8, 48)          20784     
_________________________________________________________________
batch_normalization_5 (Batch (None, 8, 8, 48)          192       
_________________________________________________________________
activation_5 (Activation)    (None, 8, 8, 48)          0         
_________________________________________________________________
global_average_pooling2d_1 ( (None, 48)                0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 48)                0         
_________________________________________________________________
dense_1 (Dense)              (None, 256)               12544     
_________________________________________________________________
p_re_lu_1 (PReLU)            (None, 256)               256       
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0         
_________________________________________________________________
predictions (Dense)          (None, 1)                 257       
=================================================================
Total params: 49,669
Trainable params: 49,381
Non-trainable params: 288
```

The total trainable parameters are 49,381 which are far less than previous two models (only ~37%). The training time is reduced significantly.

Data Augmentation

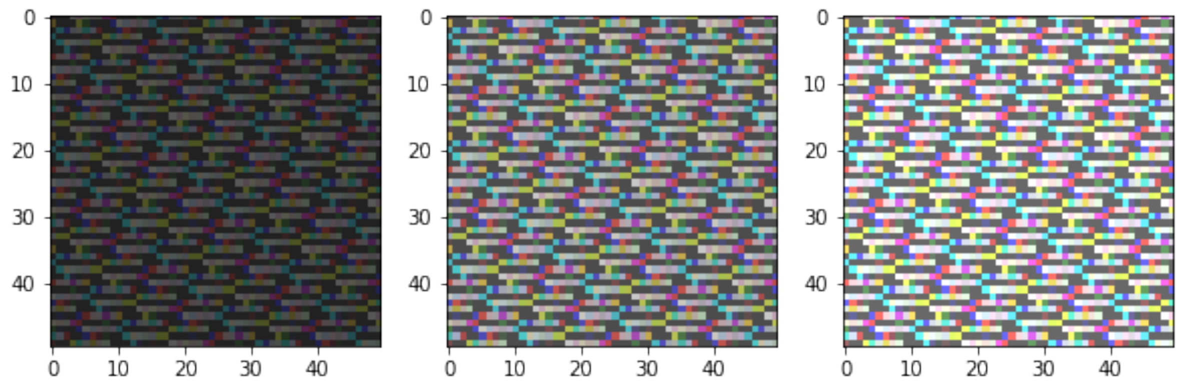
Data Augmentation, which contains a series of operations, such as adding Gaussian noise and random crop, to transform training sample into another similar but different feature, is widely used in deep learning to make model more robust. Besides, it is also a good approach to provide more training samples, especially for the deficiency of dataset.

Visualized training sample before data augmentation:

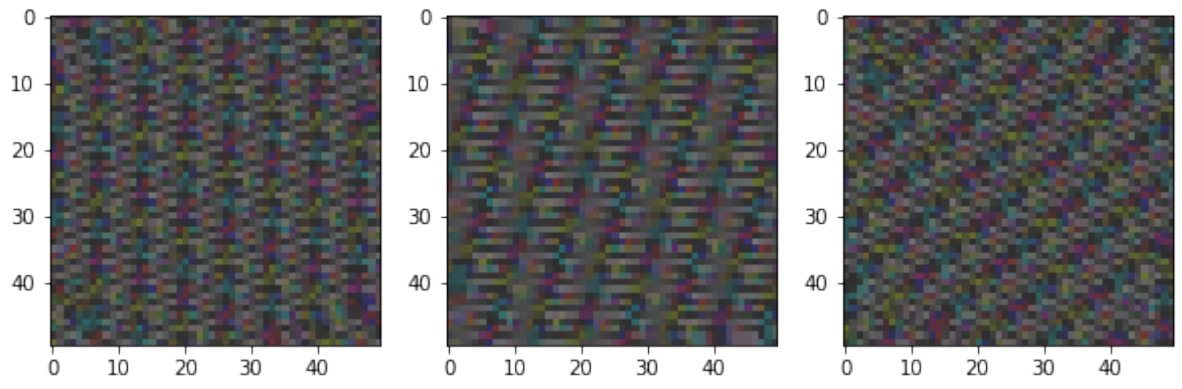


Here I try two different types of data augmentation:

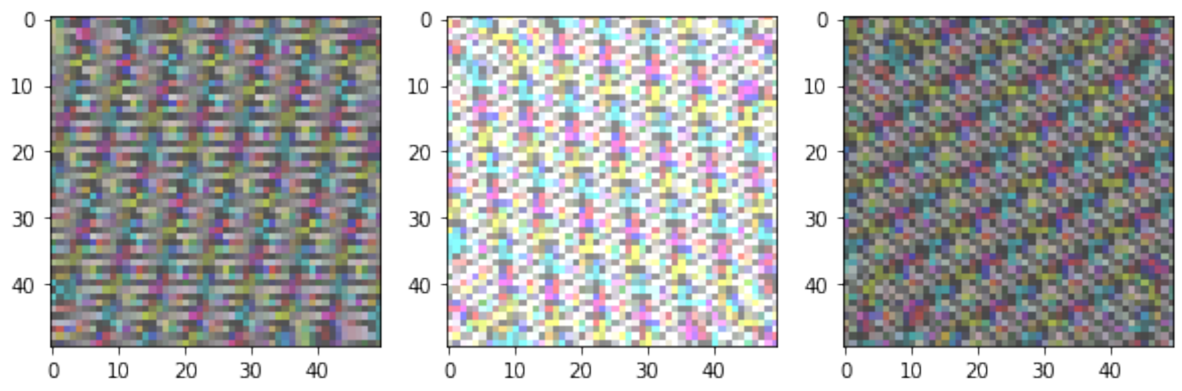
1. Random value shift: Add integer value randomly on each byte.

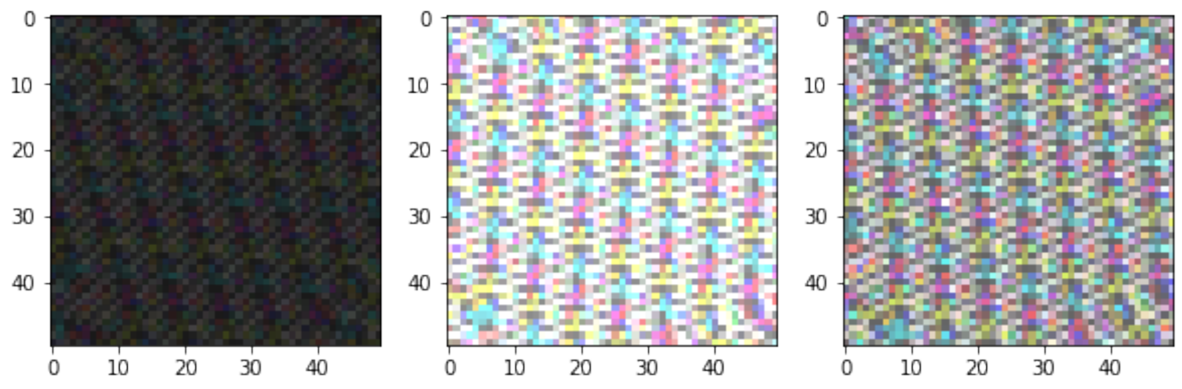


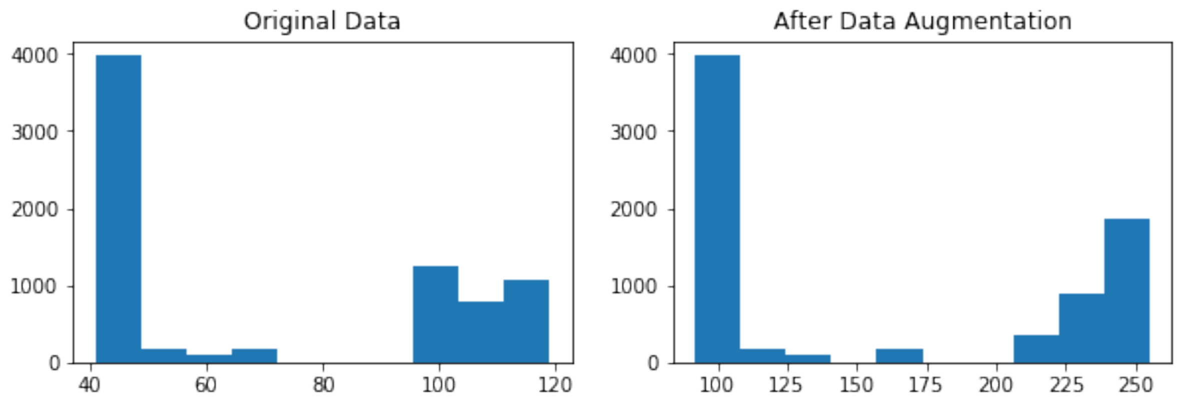
2. Data rearrangement: Rearrange the order of bytes in a package. For this project, this operation is implemented as rotation of visualized data payload.



And the combination of the above two approaches:







Histograms of the original data and the data after data augmentation.

Data augmentation operates only on training set to make data more generalized for variant changes in data. As a result, it could improve the robustness of model in most cases. For ID, data augmentation could simulate some random inserted payloads which do not appear in dataset but may exist in real world.

Evaluation Set up

This is a binary classification task. And thus, evaluation metrics consist of accuracy, precision, recall rate and false positive rate. Besides, Receiver Operating Characteristic (ROC) curve is also contained for this imbalanced dataset.

Experiment Settings

Hardwares

```
- CPU: Intel E3 1231 V3
- RAM: 16 GB
- HDD: 2 TB
- GPU: 1080Ti, 11GB VRAM
```

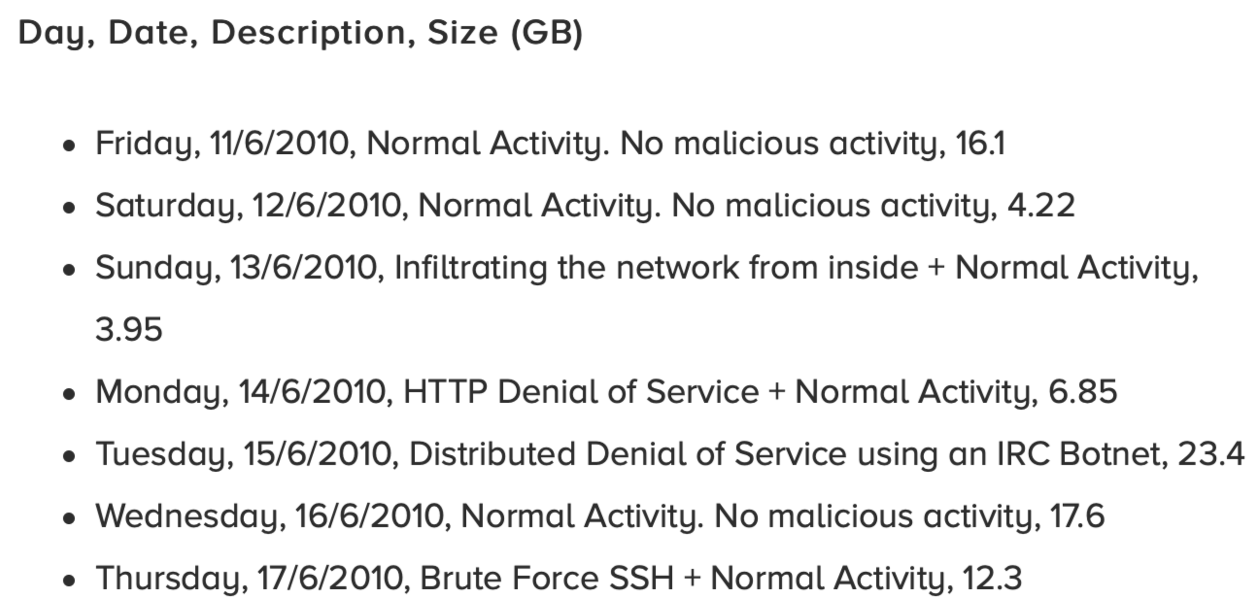
Softwares

```
- TensorFlow: 1.12.0
- Keras: 2.2.4
- Python: 3.6.8
- Jupyter-Notebook
```

ISCX IDS 2012 Dataset

https://www.unb.ca/cic/datasets/ids.html

Intrusion detection evaluation dataset (ISCXIDS2012) consists of the following 7 days of network activity (normal and malicious):



Day, Date, Description, Size (GB)

- Friday, 11/6/2010, Normal Activity. No malicious activity, 16.1
- Saturday, 12/6/2010, Normal Activity. No malicious activity, 4.22
- Sunday, 13/6/2010, Infiltrating the network from inside + Normal Activity, 3.95
- Monday, 14/6/2010, HTTP Denial of Service + Normal Activity, 6.85
- Tuesday, 15/6/2010, Distributed Denial of Service using an IRC Botnet, 23.4
- Wednesday, 16/6/2010, Normal Activity. No malicious activity, 17.6
- Thursday, 17/6/2010, Brute Force SSH + Normal Activity, 12.3

Data Preprocessing

Due to the memory limitation, I can not put all the payload into **training data**. And for this project, I take the first 7500 bytes of destination payload as data and its 'tag' in labeled xml, or normal/malicious as **training label**.

Data (Input Feature)

```python
if next_child.tag == 'destinationPayloadAsUTF':
    if next_child.text is not None:
        x = next_child.text
        if len(x) > actual:
            x = x[: actual]
        else:
            while len(x) < actual:
                x += x
            x = x[:actual]
```

To satisfy the input size requirement, payload is repeated into 7500 if it is not as long as 7500.

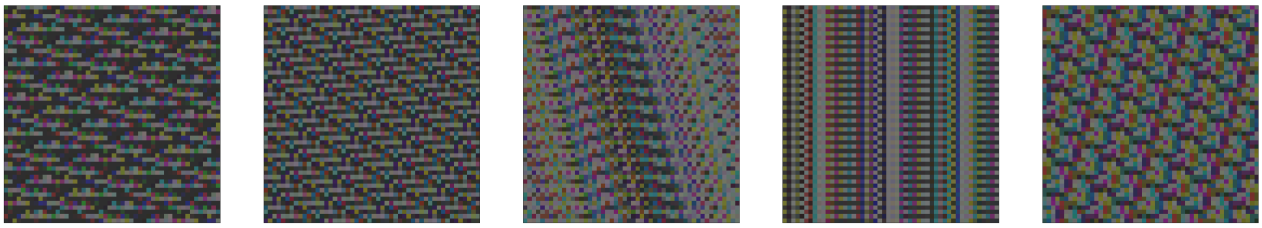
```
In [30]: child.find('Tag').text
Out[30]: 'Normal'

In [31]: next_child.text
Out[31]: '........cdn.photobucket.com.cfootprint.net......4'

In [32]: len(x)
Out[32]: 7500

In [33]: # Payload is repeated into 7500 if it is not as long as 7500
         x
Out[33]: '........cdn.photobucket.com.cfootprint.net......4........cdn.photobucket.com.cfootprint.net......4........cdn.photobucket.com.cf
         ootprint.net......4........cdn.photobucket.com.cfootprint.net......4........cdn.photobucket.com.cfootprint.net......4........cdn.phot
         obucket.com.cfootprint.net......4........cdn.photobucket.com.cfootprint.net......4........cdn.photobucket.com.cfootprint.net....
         ..4........cdn.photobucket.com.cfootprint.net......4........cdn.photobucket.com.cfootprint.net......4........cdn.photobucket.com.
         cfootprint.net......4........cdn.photobucket.com.cfootprint.net......4........cdn.photobucket.com.cfootprint.net......4........cdn.ph
         otobucket.com.cfootprint.net......4........cdn.photobucket.com.cfootprint.net......4........cdn.photobucket.com.cfootprint.net.
         .....4........cdn.photobucket.com.cfootprint.net......4........cdn.photobucket.com.cfootprint.net......4........cdn.photobucket.co
         m.cfootprint.net......4........cdn.photobucket.com.cfootprint.net......4........cdn.photobucket.com.cfootprint.net......4........cdn.
         photobucket.com.cfootprint.net......4........cdn.photobucket.com.cfootprint.net......4........cdn.photobucket.com.cfootprint.n
```

To fit the input shape of neural network, I reshape each training sample of 7500 dimensions into $50 \times 50 \times 3$ image like array. And some of such transform results are shown below.



We can find some artifacts in these data.

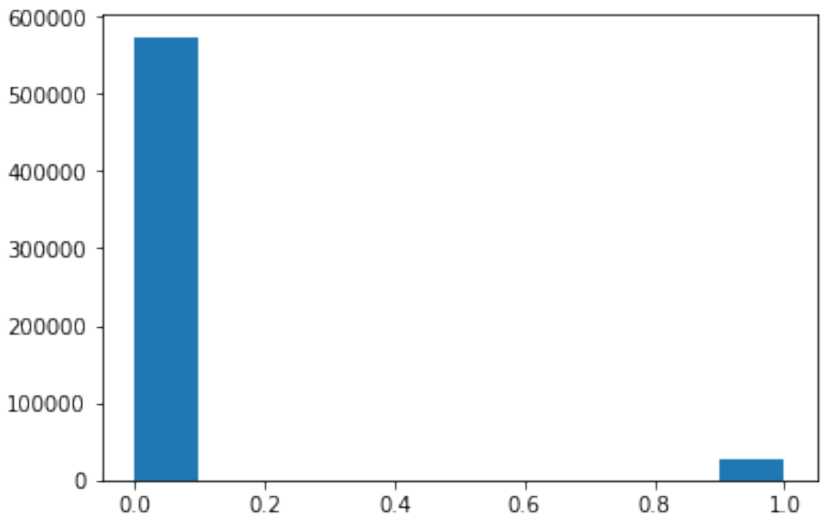
Label

```python
if child.find('Tag').text == 'Normal':
    sample_label = 0 # Normal
else:
    sample_label = 1 # Malicious
```

Label distribution:

Normal = 764712

Malicious = 38083



This is a quite imbalanced dataset with overwhelming amount of normal data, which may result into a normal-preferred model (even with high accuracy, but classifying nearly all test data into 'Normal' class). So simple showing accuracy may reflect biased result. To analyze this classification result, I plot ROC curve and calculate Area Under Curve (AUC).

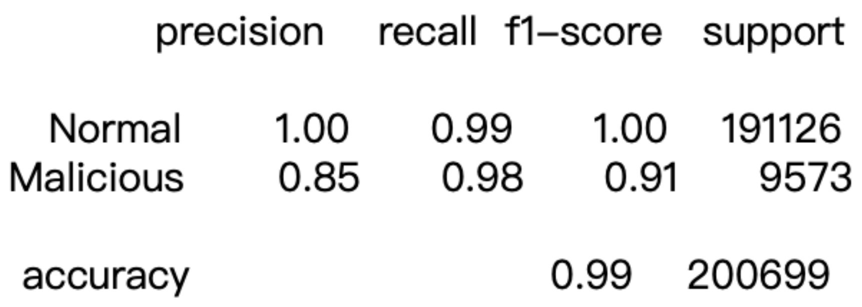
Results

All the data are split randomly into 80% training set and 20% validation set.

	Accuracy	**AUC**	**Time Cost (s/epoch)**
Prototype Model	**0.99066**	0.982	167
Model with Modified Header	0.99064	0.983	176
Light Weight Model	0.99056	**0.993**	**67**

Prototype Model

Predict Result: (Support is the number of samples)

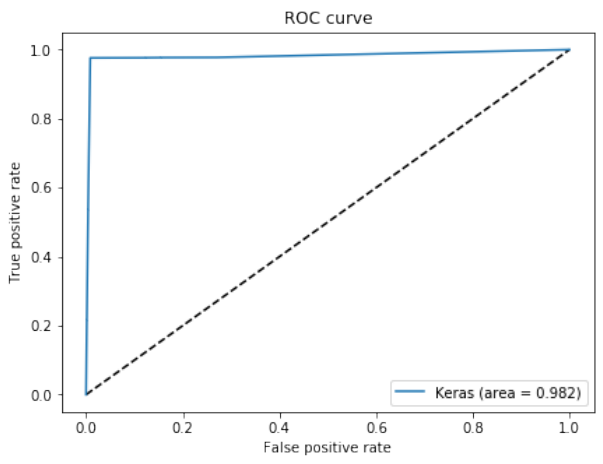


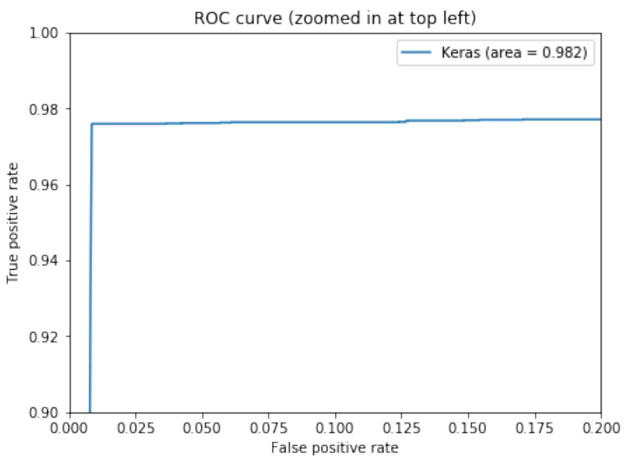
```
               precision    recall  f1-score   support

      Normal       1.00      0.99      1.00    191126
   Malicious       0.85      0.98      0.91      9573

    accuracy                           0.99    200699
```

tn(True Negative) 189481, fp(False Positive) 1645, fn(False Negative) 230, tp(True Positive) 9343

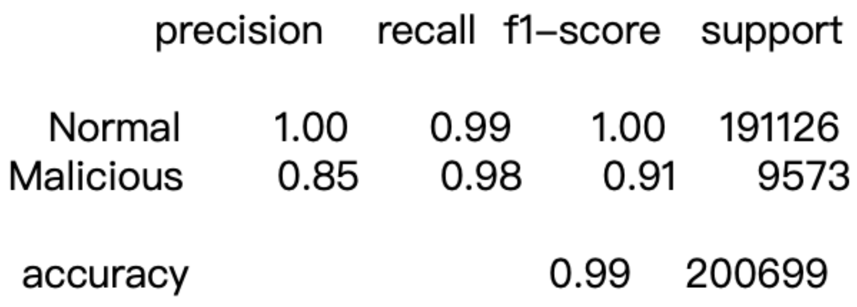




Training time: 169s/epoch

Model with Modified Header

Predict Result:

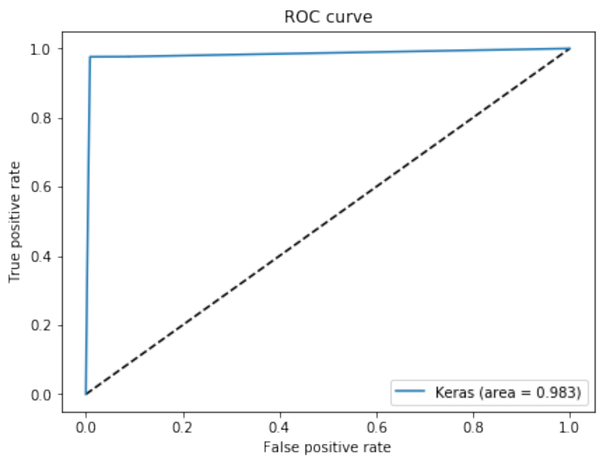


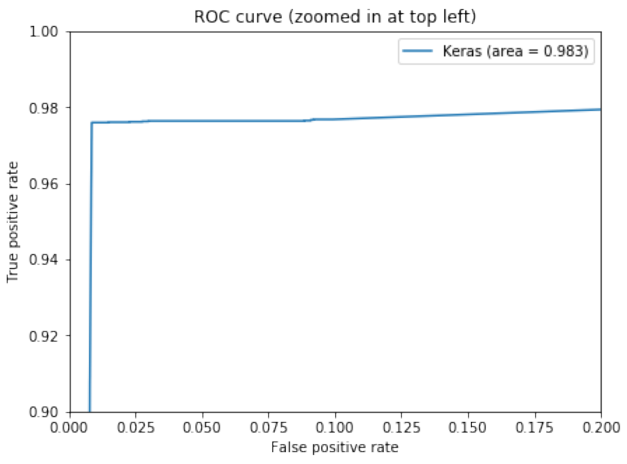
```
              precision    recall  f1-score   support

      Normal       1.00      0.99      1.00    191126
   Malicious       0.85      0.98      0.91      9573

    accuracy                           0.99    200699
```

tn: 189486, fp 1640, fn 236, tp 9337

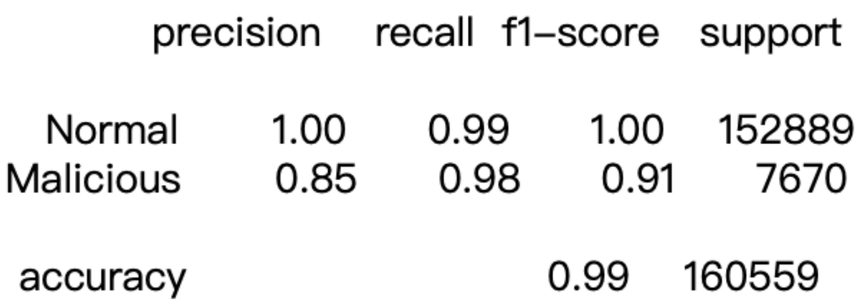




Training time: 176s/epoch

Light Weight Model

Predict Result:

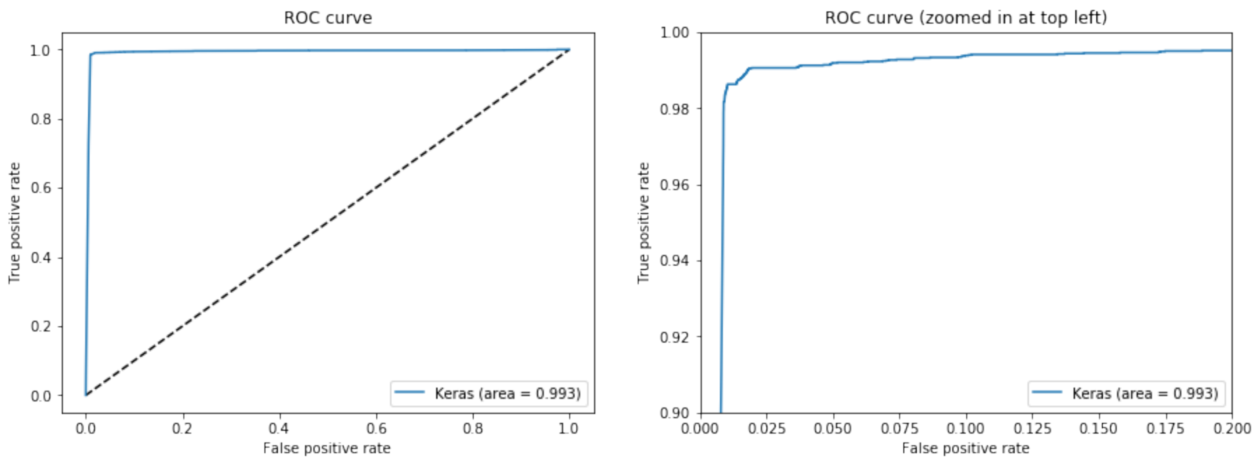


```
             precision    recall  f1-score   support

      Normal       1.00      0.99      1.00    152889
   Malicious       0.85      0.98      0.91      7670

    accuracy                           0.99    160559
```

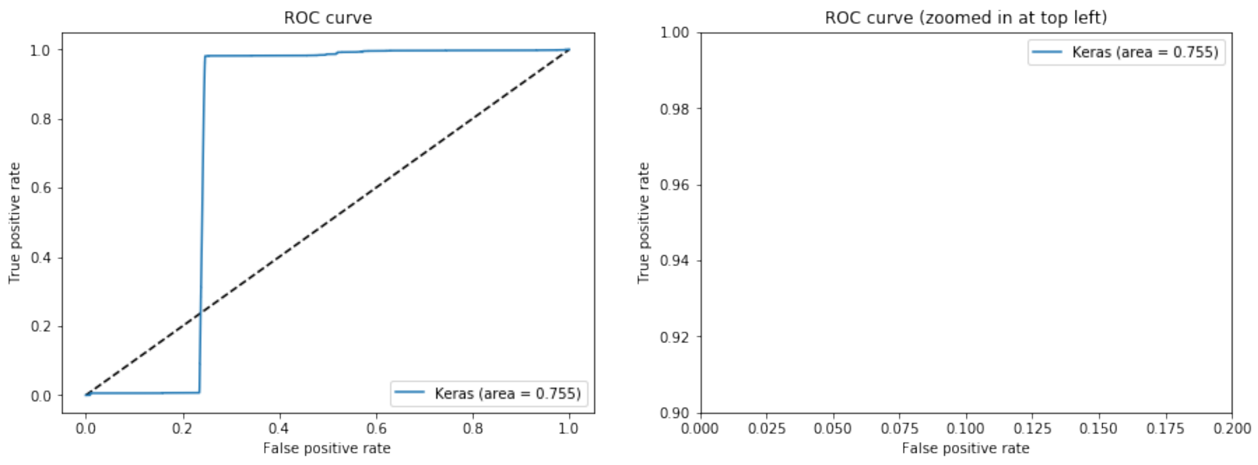
tn: 151522, fp 1367, fn 148, tp 7522



Training time: 67s/epoch

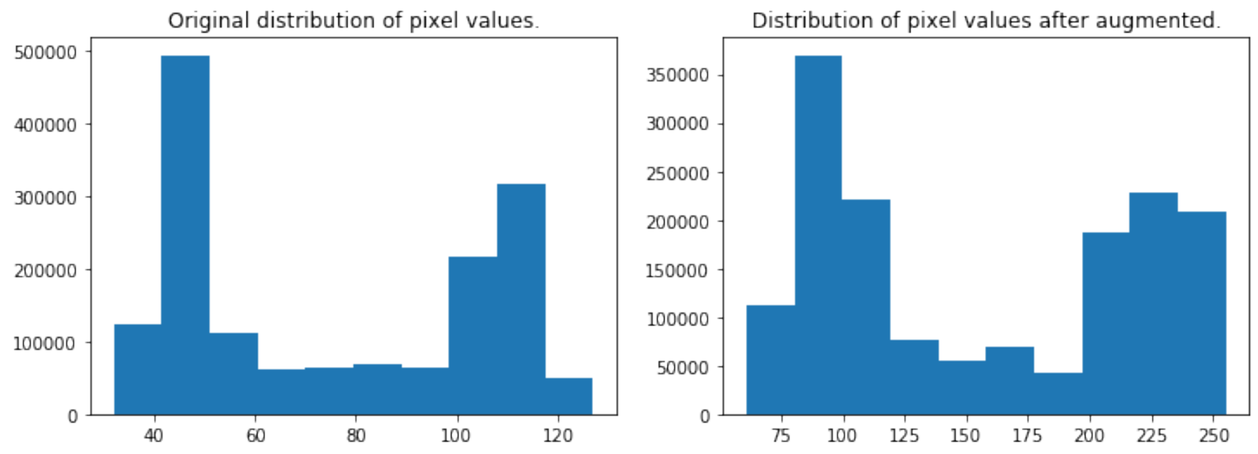
Data Augmentation

tn: 152953, fp 0, fn 7606, tp 0

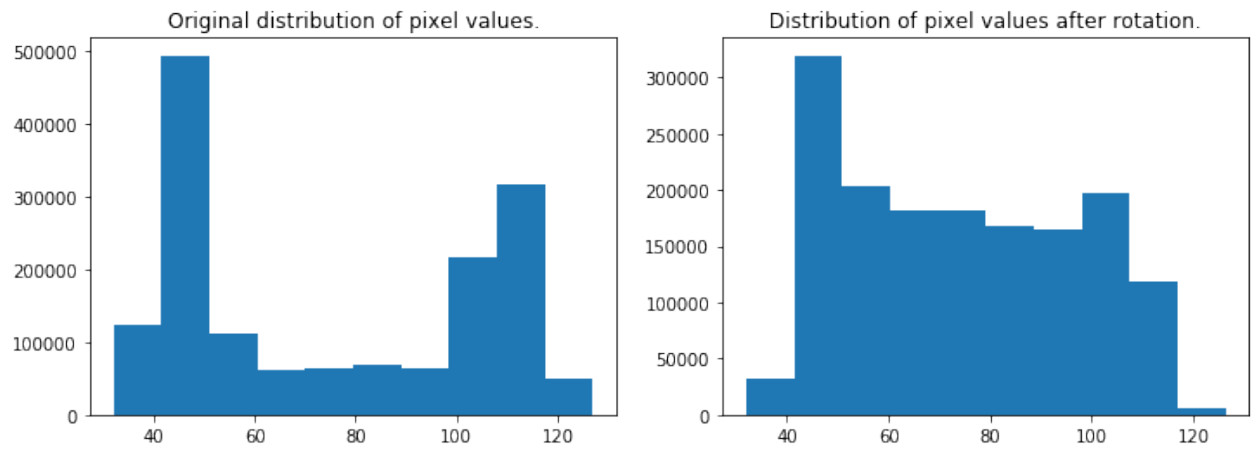


Training time: 297s/epoch

When using this procedure to train model, the result is quite disappointing. The model predicts all samples as Normal. One reason is the imbalanced label distribution (Normal ≫ Malicious). Another reason may play more important role: Data augmentation changes the distribution of data values. This phenomenon could be observed in the histograms below.



Histograms of the original data and the data after value shift.



Histograms of the original data and the data after rotation.

For random value shift, the range of data values expands. For rotation, the data values change from integers into floats. All these processing methods make training data inconsistent with validation data and lead to such poor result.

Conclusion

CNN has natural advantage in dealing with structure data and is good at extracting discriminative features, which is crucial in detection of malicious payload bytes. This project is mainly based on the settings of [3] and follows what is mentioned in Future Work part, or try different payload preprocessing method and model design. For the two introduced header, the models achieve 0.9906 accuracy but each epoch takes about 170s for training. In contrast, the introduced Light Weight Model achieves similar accuracy but only takes 67s to train each epoch. Furthermore, Light Weight Model achieves better AUC (0.993 v.s. 0.982), which is more important for such imbalanced dataset.

As for data augmentation, regular methods that work quite well in computer vision do not fit this situation. Some specific processing methods for ID are indispensable for a more robust model.

Reference

1. A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.
2. Dainotti, A., Pescape, A., Claffy, K.C.: Issues and future directions in traffic classification. IEEE Netw. 26(1), 35–40 (2012)
3. Cui, J., Long, J., Min, E., Liu, Q., & Li, Q. (2018, June). Comparative Study of CNN and RNN for Deep Learning Based Intrusion Detection System. In *International Conference on Cloud Computing and Security* (pp. 159-170). Springer, Cham.