Research on Traffic Classification of Auto Machine-learning Base on Meta-QNN

Xinyu Xu, Yongzhong Huang*, Yongcheng Luo, Guige Ouyang, Haoyu Zhuang
Guilin University Of Electronic Technology, China

*2389483289@qq.com

Abstract. The traditional traffic classification task relies on feature engineering by experts with specialized knowledge. With the advent of representation-learn-based traffic classification techniques, machines learned to extract features from traffic data and classify them. In this paper, we research traffic classification technology based on representational learning and import auto-machine learning technology to solve the problems of network architecture design and parameter tuning. We also design a befitting reward function for the network architecture model. The experimental results on the USTC-TF2016 dataset and USTC-TF2016-PLUS shows that the network architecture generated by auto-machine learning technology has better training performance and classification accuracy than a traditional neural network.

Keywords: Network security, Traffic classification, Convolution neural network, Auto machine learning, Network architecture search, Reinforcement learning

1. Introduction
As an effective means of network perception, network traffic detection has attracted more and more attention from researchers in recent years. So far, scholars at home and abroad have proposed many different types of detection methods. Among them, technology based on network traffic classification is a significant one.

At present, the mainstream traffic classification methods include the following five categories: port-based, deep-packet inspection (DPI), statistics-based, behavior-based, and presentation-based learning [1]. Among them, the methods based on port and deep packet inspection are rule-based classification methods. By designing specific matching rules, the network traffic can be hard-coded matching classification. Statistical-based and behavior-based classification methods are traditional machine learning methods that classify traffic by extracting patterns from empirical data using a set of chosen features. The biggest drawback of these conventional methods is that they rely on expert knowledge for feature engineering. The classification method based on representation learning automatically learns features from the original data, which solves the problem of manual design features to some extent, still needs someone to tuning parameter.

The traffic classification method based on representation learning can extract the traffic characteristics of various network traffic from the dataset. But for different datasets, different network architectures, or multiple hyper-parameter choices under the same network architecture, the network will also have variant performance. How to design the right network architecture, or how to adjust the hyper-parameters based on the existing excellent network structure, are the accompanying problems.
The appearance of Automated Machine Learning (AutoML) provides a solution to the problem. AutoML aims to automate the processes of machine learning construction, feature engineering, model selection, parameter tuning, model evaluation, etc., reduce or even eliminate manual intervention in the process of model construction, and achieve good generalization performance and saving training time. Since its inception, AutoML has attracted extensive attention from academia and industry and generated a large amount of research work and commercial products. Neural Architecture Search (NAS), which is an efficient and concise network architecture search idea, is also one of the bases of today's mainstream AutoML thought. Its feature is to find the most suitable network architecture by training different combinations of each heap block to build the network model. In this paper, automatic machine learning technology is introduced into the representation-learning-based traffic classification system and tested on the open dataset USTC-TFC2016. To contrast with the effect of the representation-learning-based traffic classification method, we expand 15 additional malicious traffic data using the same data preprocessing method of the USTC-TF2016 dataset and mix them with it, making them a new dataset USTF-TF2016-PLUS. And we use this new dataset to test the Meta-QNN base NAS model's performance with the representation-learning-based classification model.

The rest of this paper's structure as follows: The second section describes related work and introduces our motivation; The third section includes our model's detail; The experiment's result can be seen in section four. And section five describes something about our current work and the future.

2. Related work
The rule-based traffic classification method is relatively mature in the industry, and related work mainly focuses on how to accurately extract mapping rules to improve performance. Finsterbusch et al. [2] summarized the current main traffic classification methods based on DPI. The classical traffic classification model based on machine learning has attracted a lot of academic research, and the related work mainly focuses on how to choose a better feature set. Dhote et al. summarized feature selection techniques for Internet traffic classification. Blake Anderson et al. [3] classified the malicious traffic over HTTP and HTTPS. Through the parsing of HTTP/HTTPS packets, they automatically found the important field characteristics in the packets and used them as the classification basis for the training network.

Traffic classification based on representational learning is also a hot research topic at present. Gao et al. [4] proposed: Classification method of malware traffic based on a deep belief network; Javaid et al. [5] proposed a malicious traffic identification method based on sparse self-coding. Zhang Hui et al. [6] used CNN multiple classifiers to detect malicious URLs. Wang Wei et al. [7] proposed traffic classification based on convolution neural network representational learning. The greatest advantage of representation-learning lies in its ability to learn features from raw data, which is also the reason why deep learning has achieved significant results in image classification and speech recognition. But they work on specific datasets, designing or using designated networks. For example, Gao et al. use the deep belief network in their paper; Javaid et al. use an autoencoder network for their work; The convolution unique neural network designed by Zhang Hui et al.; And the LaNet-5 convolution neural network used by Wang Wei et al. The disadvantage of a specific network structure is: if there is a different dataset used in the task, the classification network may not do great.

Network architecture search is an emerging technology, which is characterized by that for different data set tasks, the model can find a suitable classification network in the exploration space to improve the performance of this problem. BowenBaker et al. [8] first proposed using reinforcement learning to explore CNN network architecture in 2016. In the same year, GoogleBrain also published a paper [9], using reinforcement learning to train an RNN network and generate CNN network architecture. In 2018, GoogleBrain improved the model and proposed NAS-NET architecture [10]. This year, GoogleBrain implemented a pyramidal web search model called NAS-FPN[11]. Domestic research is inclined to the application of AutoML technology, such as Liu Guixiong et al. [15] used AutoML technology for semantic segmentation; Liang Qingqing[16] realized automatic super-parameter optimization; and Baidu's PaddlePaddle platform combined THE AutoML technology based on PG
algorithm into the visual ARTIFICIAL intelligence framework, reducing the cost of artificial learning and development.

Inspired by these studies, this paper introduces automatic machine learning technology into network traffic classification tasks. The method used in this paper is the Neural network Search Architecture based on Q-learning. This classical reinforcement learning algorithm is suitable for the environment with limited behavioral space. It performs well on the task of Neural Architecture Search (NAS) and has lower performance requirements than other Search algorithms.

3. Model structure

Convolution neural network (CNN) is widely used in various classification problems and has achieved excellent results in various fields. Researchers have also designed different structures of convolution neural networks for different problems and production environments. NAS technology is an optimization scheme proposed for the structural design of neural networks, allowing computers to design their networks. The model in this paper is based on the chain network search architecture in NAS.

3.1. Dataset

The dataset selected in this paper is the USTC-TFC2016 data set [7]. This data set is composed of 10 kinds of malicious traffic selected from the CTU data set [17] (Tab 3.1) and 10 kinds of normal traffic newly collected using the LXIA BPS device (Tab 3.2). Also, according to the work of the paper [7] and according to its preprocessing mode, 15 additional types of original PCAP packets of malicious traffic were selected from CTU data and processed into the USTC-TFC2016-PLUS data set (Tab 3.3) as the comparative experimental data.

The data preprocessing process can be divided into four steps: traffic segmentation, traffic cleaning, image generation, and IDX conversion. The complete work flow chart is shown in figure 1. This process refers to the work of [12] and processes packets into four different types, Flow + All, Flow + L7, Session + All, and Session + L7. It also USES the method of anonymous [13] to eliminate the IP or MAC segment characteristics of the traffic.

| Name  | CTU num | Binary MD5               | Process     |
|-------|---------|--------------------------|-------------|
| Cridex | 108-1   | 25b8631afeea279ac00b2da70ffe18a | Original    |
| Geodo  | 119-2   | 036573e52008779a0801a25f9b18101 | Part        |
| Htbot  | 110-1   | e515267ba19417974a63b1e4f7dd9e99 | Original    |
| Miuref | 127-1   | a41d395286deb113e17bd3f4b69ece182 | Original    |
| Neris  | 42,43   | bf08e6b020e0d2bc6dd493e9369872f | Merged      |
| Nsis-ay| 53      | ea85db9898d3c9101d5fca44c630e4 | original    |
| Shifu  | 142-1   | b9bc3f1b2aae824482c10effa422f78b | part        |
| Tinba  | 150-1   | e9718e38e35e331e6be281e4ecfae8 | part        |
| Virut  | 54      | 85f9a5247fabe51e6479419f3fd72eb | original    |
| Zeus   | 116-2   | 8df6603d7cbe2fd5862b14377582d46 | original    |

Tab.1 Ustc-Tfc2016 Part I (Malware Traffic)

| Name   | Class       | Name        | Class       |
|--------|-------------|-------------|-------------|
| BitTorrent | P2P         | Outlook     | Email/WebMail|
| Facetime | Voice/Video | Skype       | Chat/IM     |
| FTP     | Data Transfer | SMB        | Data Transfer|
| Gmail   | Email/WebMail | Weibo  | Social Network|
| MySQL  | Database    | World Of Warcraft | Game   |

Tab.2 USTC-TFC2016 PART II (NORMAL TRAFFIC)
3.2. Model design

CNN can be viewed as an integral structure makeup by multi-network layers. And the process of network architecture searching can be treated as exploring a maze with many fork road. An agent's status change in the maze is a complete Markov process, so we can use reinforcement learning to let the agent learn to build a network.

In this paper, we adopt a single chain-like NAS model based on Meta-QNN [8] and adjust search space and search rules suitable for this task. We will explain the model of this project from three aspects: searching space, explore policy, evaluation method, and the NAS model based on Q-Learning.

3.2.1. Searching Space

CNN network is composed of three different network layers: convolution layer, pooling layer, and dense layer. The convolution layer has three hyperparameters: convolution kernel (Kernel size), stride length (strides), and output dimensions (Filters). The pooling layer has two hyperparameters: pool size, strides. And the dense layer has only one hyperparameter: output dim. For each hyperparameter, there is a range of values that can be selected. Tab 3.4 shows all those value ranges.

We define that: 1. Any agent can explore and build a network with no more than 12 layers. 2. Any agent has the liberty to stop exploring at any time step. 3. When an agent stops exploring, a terminal layer, which is constituted by a dropout layer and a softmax layer, will be added into the network this agent had explored and built.

Besides, to reduce the parameter scale of the training network, we stipulate that only the layer with an output dimension less than eight can choose a dense layer to connect with as the following layer, and the total dense layer's amount can't exceed three.
3.2.2. Explore Policy
In this task, $\varepsilon$-greedy policy is used in this paper, which is characterized by a complete greedy strategy or random selection strategy determined by probability under the control of parameters. At the beginning of the task, the parameters are set to a small value. In this way, at the beginning of the task, individuals tend to explore randomly to find more unknown state values. The parameter will increase slowly with the exploration round, and the maximum increase will be 0.9 so that the random exploration can be guaranteed even if the value is stable in the later period. This parameter setting is as follows:

We use $\varepsilon$-greedy policy, whose characterize is using the parameter $\varepsilon$ to determine the probability of complete greedy policy or random selection policy. $\varepsilon$ is set as a small value at the beginning of a task, and will slowly grow up during experience. That can make the agent tend to explore randomly to find more unknown status the very first and convergence in the later period of experience. Tab 3.5 shows the parameter $\varepsilon$ for different training rounds.

| Layer parameters | Search space |
|------------------|--------------|
| convolution Kernel size | Int(range(1,6)) |
| Strides | Int(range(1,2)) |
| Filters | Int(range(1,5))*64 |
| Pool Pool size | [2, 3, 5] |
| Strides | [2, 3] |
| Dense Dense size | Int(range(1,5))*64 |

| Model trained(1e1) | $\varepsilon$ |
|---------------------|----------------|
| 100 | 0.0 |
| 10 | 0.1 |
| 10 | 0.2 |
| 10 | 0.3 |
| 15 | 0.4 |
| 15 | 0.5 |
| 15 | 0.6 |
| 15 | 0.7 |
| 15 | 0.8 |
| 15 | 0.9 |

Tab.4 Network layers Search space

Tab.5 $\varepsilon$ for training rounds

The advantage of $\varepsilon$-greedy policy is that it can not only make agents explore enough states, but also make the value of the exploration converge in the later period.

3.2.3. Evaluation Method
For the generation of network structure, this paper designs an appropriate evaluation method. Since a network model can only conduct data training after the combination is completed, this paper stipulates that the reward for each step experienced by the agent in the experience is 0. Only when the individual reaches the terminal state, the environment will conduct training evaluation on the convolution network model formed by the agent according to the experience. The assessment function selected in this paper is the accuracy function, and the calculation formula of the accuracy algorithm is:

$$\frac{TP + TN}{TP + TN + FP + FN}$$

In TensorFlow, four evaluation values are returned after the network model training, $[\text{loss, acc, loss test, acc test}]$, respectively representing the loss and accuracy of the model on the training set. Loss and accuracy on the testing set. We hope to find a model that can make its accuracy high on the training set and the test set, loss as low as possible, and the difference between the accuracy on the two data sets as small as possible (generalization ability), so we design the formula as follows:
\[
\text{reward} = \alpha(\text{acc} \times \text{acc}_\text{val}) \times e^{-\text{(loss \times loss}_\text{val})} - \beta(\text{acc} - \text{acc}_\text{val})^2
\]  

(2)

If we consider \([\text{loss}, \text{loss}_\text{val}]\) as \(x\) set, while \([\text{acc}, \text{acc}_\text{val}]\) as \(y\) set, formula (2) can be written as:

\[
F_{\text{reward}} = \alpha F_1(x, y) - \beta F_2(y)
\]

(3)

In formula (3), \(F_1\) represents the evaluation score of the model, while \(F_2\) represents the generalization loss of the model. \(\alpha, \beta\) are those function's weight.

Because of \([\text{loss}, \text{acc}, \text{loss}_\text{val}, \text{acc}_\text{val}]\) are all in the range \([0, 1]\), so the maximum point of \(F_1\) will be in \(x=[0, 0], y=[1, 1]\), and the minimum point of \(F_2\) will be in \(\text{acc} = \text{acc}_\text{val}\) set.

If During the experiment, we found that \(\alpha, \beta\) are important for the NAS model. The agent will convergence to an overfitting network if \(\alpha\) is too big, or convergence to a low accuracy network with a low overfitting score when \(\beta\) is too big. Tab 3.6 lists results with the different \(\alpha, \beta\) value used for training the MNIST dataset, according to which we decide to use \(\alpha \in (0, 0.2], \beta \in [0.8, 1)\) for the subsequent experiment.

| \((\alpha, \beta)\) | Accuracy_val | Loss_val | Network architecture* |
|------------------|--------------|----------|-----------------------|
| (0.9,0.1)        | 0.993        | 0.172    | c(3,1,64);c(3,1,128);p(3,2);c(3,1,192);p(3,2);c(5,1,64);p(5,3);d(0.5);s(10) |
| (0.8,0.2)        | 0.994        | 0.165    | c(5,1,192);p(3,2);c(3,2,64);p(3,2);c(5,1,128);p(3,3);d(0.5);s(10) |
| (0.7,0.3)        | 0.982        | 0.121    | c(1,1,128);c(3,1,64);c(3,1,128);p(5,2);c(5,2,128);p(3,2);d(0.5);s(10) |
| (0.6,0.4)        | 0.978        | 0.086    | p(3,2);c(3,1,64);c(3,1,64);p(5,2);d(0.5);s(10) |
| (0.5,0.5)        | 0.950        | 0.128    | p(3,2);c(5,2,320);p(5,3);c(3,1,192);p(3,2);c(3,2,192);c(3,1,64);c(3,1,64);d(0.5);s(10) |
| (0.1,0.9)        | 0.837        | 0.023    | c(3,1,192);p(3,2);c(5,2,320);c(3,2,64);d(0.5);s(10) |

Tab.6 Result of different combination
*(c: conv layer; p: pool layer; d: dense layer; s: dense layer with different output dim)

3.2.4. NAS model base on Q-learning

Q-learning is one of the Value-based algorithms of Reinforcement Learning (RL). The objective of the Q-Learning algorithm is to find an appropriate strategy that enables the Agent to make the optimal choice of action when it is aware of its status [14]. It does not require an environment model and can handle random transformations and rewards without adjustment.

Suppose that we now have an Env (environment), with an agent in it. The agent’s position, individual state, and observable environment variables are called S (status). The behavior that the agent can choose under the current state is called A (action), and the environment will give the agent R (reward) when the agent acts in Env for each step.

The ultimate goal of Q-learning is to get the optimal Q(s, a) function, which means to find the best behavior under state S. At the time t, the action the agent use to interact with the environment generated by policy \(\mu\):
\[ A_t = \mu(S_t) \]  

(4)

The policy \( \mu \) is a \( \varepsilon \)-greedy strategy. The action \( A_{t+1} \) used to update value Q at time \( t+1 \) is shown as follow:

\[ A'_{t+1} \sim \pi(S_{t+1}) \]  

(5)

The policy \( \pi \) is completely greedy. We update \( Q(S_t, A_t) \) according to the following formula:

\[ Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)) \]  

(6)

In this formula, \( \alpha \) is the learning rate, and \( \gamma \) is the decay factor. According to this update method, the value of action \( A_t \), which was obtained in state \( A_t \) according to \( \varepsilon \)-greedy policy, will be updated to the direction of the maximum behavior value determined by the greedy strategy in \( S_{t+1} \) state in a certain proportion. This algorithm makes the agent’s behavior policy \( \mu \) more similar to the policy \( \pi \), makes sure that the agent can continuously explore and experience enough new states, and finally converge to the optimal policy and the optimal behavior value function.

The complete Q-Learning value update formula as follows:

\[ Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)) \]  

(7)

And the pseudo-code of the Q-Learning algorithm is shown as algorithm 1.

### Algorithm 1 Q-learning algorithm

**Input:** episodes, \( \alpha \), \( \lambda \)  
**Output:** \( Q \)

1. initialize: \( Q(s, a) \) arbitrarily, for each \( s \) in \( S \) and \( a \) in \( A(s) \); set \( Q(\text{terminal state, } *)=0 \)
2. for each \( \text{episode} \in \text{episodes} \) do
   3. initialize: \( S \leftarrow \text{first state of episode} \)
   4. for each \( \text{step} \in \text{episode} \) do
      5. \( A = \text{policy}(Q, S) \)
      6. \( R, S' = \text{perform action}(S, A) \)
      7. \( Q(S, A) \leftarrow Q(S, A) + \alpha(R + \lambda \max_a Q(S', a) - Q(S, A)) \)
      8. \( S \leftarrow S' \)
      9. if \( S \) is terminal state then break
   10. end if
   11. end for
12. end for

4. Experimental results and analysis

4.1. Model training
In this paper, we define the agent's learning rate $\alpha$ as 0.4, and the agent's attenuation coefficient $\gamma$ as 1. $\gamma$ set to 1 to enable the agent to pursue future value as far as possible. To reduce the total search time, we set the number of epochs as 100 training rounds for each constructed network. Besides, we stipulate that when the agent explores the repeated network architecture, it will re-train that network with a probability of $P=0.4$, otherwise read the previous training scores from memory. Also, this model sets exit conditions. If the network structure, whose evaluation score exceeds the score we define (the required standard score), is found, the model will output this network structure and exits.

Figure 2 shows the round-evaluation score of the model on the USTC-TFC 2016 dataset. We found that whether it is the LaNet-5 network architecture or the network architecture searched by our model, there is no significant difference between 10 classification tasks and 20 classification tasks in the original USTC-TFC2016 dataset, both accuracy rate can reach above 0.98. So we expanded the malware traffic data of USTC-TFC2016 into USTC-TFC2016-PLUS and made them contain different kinds of malware traffic types from 10 to 25. Then we randomly selected 10 of them and trained with the LaNet-5 CNN network. Figure 3 shows the result of the LaNet-5 model training with ten random kinds of malware traffic data.

![Fig.2 Training with USTC-TF2016](image)

We can see there are some phenomenon of overfitting occurs in figure three. And we found that SessionL7 performs better than other pretreatments method, so we choice SessionL7 as our primary pretreatment method in the subsequent experiments. Furthermore, we also add the second early stopping method to avoid overfitting. Figure four shows the training result from 10 to 25 classification using LeNet-5 CNN network.

We use our network architecture search model for 10 to 25 classification tasks, tab 4.1 lists our result compare with LeNet-5, and tab 4.2 lists the network architecture we create.

4.2. Result analyze

We found that for the original USTC-TFC2016 dataset, the LeNet-5 model can meet the user demand, but it performs poorly when the dataset expands or changes. It means a single CNN network may not suitable for complex net network traffic datasets. At the same time, the NAS model can find a sort of network architecture according to the different datasets and improve classification accuracy.
Fig. 4. Training result from 10 to 25 classification use LeNet-5

| Classify types | Accuracy(lenet-5) | Accuracy(ours) | Classify types | Accuracy(lenet-5) | Accuracy(ours) |
|----------------|------------------|----------------|----------------|------------------|----------------|
| 10             | 0.9782           | **0.9791**     | 18             | 0.9713           | **0.9750**     |
| 11             | 0.9801           | **0.9803**     | 19             | 0.9721           | **0.9743**     |
| 12             | 0.9791           | **0.9801**     | 20             | 0.9726           | **0.9782**     |
| 13             | 0.9781           | **0.9787**     | 21             | 0.9743           | **0.9760**     |
| 14             | 0.9790           | **0.9801**     | 22             | 0.9758           | **0.9764**     |
| 15             | 0.9629           | **0.9781**     | 23             | 0.9683           | **0.9701**     |
| 16             | 0.9611           | **0.9762**     | 24             | 0.9674           | **0.9785**     |
| 17             | 0.9518           | **0.9772**     | 25             | 0.9696           | **0.9788**     |

Tab. 7. 10-25 classification tasks result

| Classify types | Network architecture* |
|----------------|-----------------------|
| 10             | c(5,1,128);c(3,1,128);p(3,2);c(5,1,192);c(1,1,128);p(3,2);c(5,1,64);p(5,3);d(0.5);s(10) |
| 11             | c(1,1,128);c(3,1,256);p(3,2);c(5,1,128);p(3,2);c(5,1,64);d(0.5);s(12) |
| 12             | c(5,1,64);c(3,1,192);d(0.5);s(12) |
| 13             | c(5,1,256);c(5,1,192);c(3,1,256);p(2,2);c(1,1,128);p(3,2);c(5,1,64);d(0.5);s(13) |
| 14             | c(1,1,256);c(5,1,192);c(3,1,256);p(2,2);d(0.5);s(14) |
| 15             | c(3,1,256);p(5,3);c(5,1,256);c(3,1,64);p(2,2);d(0.5);s(15) |
| 16             | c(5,1,64);c(3,1,64);d(0.5);s(16) |
| 17             | c(3,1,128);c(3,1,256);d(0.5);s(17) |
| 18             | c(3,1,192);c(5,1,64);c(5,1,128);d(0.5);s(18) |
| 19             | c(3,1,64);c(5,1,64);d(0.5);s(19) |
| 20             | c(5,1,64);c(1,1,128);c(3,1,192);c(3,1,192);d(0.5);s(20) |
| 21             | c(5,1,128);c(1,1,192);d(0.5);s(21) |
| 22             | c(3,1,192);c(5,1,128);p(5,3);c(5,1,192);d(0.5);s(22) |
| 23             | c(3,1,128);c(3,1,192);p(3,2);c(5,1,192);d(0.5);s(23) |
| 24             | c(3,1,192);c(5,1,192);p(5,3);d(0.5);s(24) |
| 25             | c(3,1,192);c(5,1,192);p(5,3);d(0.5);s(25) |

Tab. 8. Network architectures for 10-25 classification tasks

*(c: conv layer; p: pool layer; d: dense layer; s: dense layer with different output dim)

5. Conclusion
In this paper, we attempted to use NAS technology for malware network traffic classification and compare its effect with the LeNet-5 CNN model. The experiment shows that different net structures can curse different results, and the NAS model can find a batter net architecture in complex classification tasks. Although Meta-QNN is a simple NAS model, whose performance is not as good as Google's NAS model [9], its architecture search process is more simple and can be run at a poor hardware environment with a nice optimize. The NAS model with DQN and PG arithmetic use for net network traffic classification will be our goal in the future.

Acknowledgments
This work was supported by the National Natural Science Foundation of China under grant no. 61866008 and Basic and Applied Basic Research Fund of Guangdong Province, China under grant no. 2019B1515120085.

References
[1] Dhote Y and Agrawal S 2015 A Survey on Feature Selection Techniques for Internet Traffic Classification *Computational Intelligence and Communication Networks* (Jabalpur India) pp 1375-1380
[2] Finsterbusch M, Richter C, Rocha E, Muller J A and Hanssgen K 2014 A Survey of Payload-Based Traffic Classification *J. IEEE Approaches Communications Surveys & Tutorials* 16(2) pp 1135-1156
[3] Blake A, Andrew C, Scott D and David M 2018 Limitless HTTP in an HTTPS World: Inferring the Semantics of the HTTPS Protocol without Decryption *Preprint* arXiv:1805.11544
[4] Gao N, Gao L and Gao Q 2014 An Intrusion Detection Model Based on Deep Belief Networks *Advanced Cloud and Big Data 2014 2nd Int Conf* (Huangshan, Anhui) pp 247-252
[5] Javaid A Y, Niyaz Q, Sun W and Alam M 2015 A Deep Learning Approach for Network Intrusion Detection System *Proc.9th EAI Int Conf on Bio-inspired Information and Communications Technologies* (New York)
[6] Hui Z, Liping Q and Wangli D 2019 Detection of malicious URLs based on CNN and multiple classifiers *Computer engineering and design* 40(10) pp 2991-2995
[7] Wei W, Ming Z, Xuewen Z, Xiaozhou Y and Yiqiang S 2017 Malware traffic classification using convolution neural network for representation learning 2017 *Int Conf on Information Networking* (Da Nang) pp 712-717
[8] Bowen B, Otkrist G, Nikhil N and Ramesh R 2017 Designing Neural Network Architectures using Reinforcement Learning 2017 *ICLR* (Toulon, France) https://openreview.net/forum?id=S1c2cvqee
[9] Barret Z, and Quoc V L 2017 Neural Architecture Search with Reinforcement Learning 2017 *ICLR* (Toulon, France) https://openreview.net/forum?id=r1Ue8Hcxg
[10] Barret Z, Vijay V, Jonathon S and Quoc V L 2018 Learning Transferable Architectures for Scalable Image Recognition 2018 *Conf on Computer Vision and Pattern Recognition* (Salt Lake City: IEEE) pp 8697-8710
[11] Golnaz G, Tsung Y L, Ruoming P and Quoc V L 2019 NAS-FPN: Learning Scalable Feature Pyramid Architecture for Object Detection 2019 *Conf on Computer Vision and Pattern Recognition* (Long Beach, CA, USA:IEEE) pp 7036-7045
[12] Wang Z 2015 The Applications of Deep Learning on Traffic Identification *Blackhat USA 2015* https://goo.gl/WoulM6
[13] Koukis D, Antonatos S, Antoniades D, Markatos E P and Trimintzios P 2006 A Generic Anonymization Framework for Network Traffic 2006 *IEEE Int Conf on Communications* (Istanbul) pp 2302-2309
[14] Jin C, Allen-Zhu Z, Bubeck S and Jordan M I 2018 Is Q-learning provably efficient? *Advances in Neural Information Proc Systems* (Montral, Canada) pp 4868—4878
[15] Guixiong L, Jian H, Siyang L and Pu L 2019 AutoML method for semantic segmentation of
machine vision *J. Laser magazine* **40** (06) pp 1-9

[16] Qingqing L 2019 Performance optimization of supervised AutoML based on key super parameter selection *Guizhou University*

[17] Sebastian G, Martin G, Jan S and Alejandro Z 2014 An empirical comparison of botnet detection methods *J.Computers and Security Journal, Elsevier* **45** pp 100-123