Deep Convolutional Neural Networks for Log Event Classification on Distributed Cluster Systems

Conference Paper · December 2018

DOI: 10.1109/BigData.2018.8622611

Deep Convolutional Neural Networks for Log Event Classification on Distributed Cluster Systems

Rui Ren*†, Jiechao Cheng‡, Yan Yin*, Jianfeng Zhan*,Lei Wang*
*State Key Laboratory of Computer Architecture, Institute of Computing Technology, CAS, Beijing, China
†University of Chinese Academy of Sciences, Beijing, China
‡Xiaomi Technology Co., Ltd., Beijing, China
Email:{renrui, zhanjianfeng, wanglei_2011}@ict.ac.cn, jetrobert19@gmail.com, yinyan512@foxmail.com

***Abstract*—With the widespread development of cloud computing, cluster systems are becoming increasingly complex, and analyzing system logs is an universal and effective approach for automatic system management and troubleshooting. Log event classification as an effective preprocessing method for log analysis, which is helpful for system administrators to locate or predict which components/services have errors or failures.**

In this paper, we design and implement an automatic log classification system based on deep CNN (Convolutional Neural Network) models, and take advantage of the feature engineering and learning algorithm to improve classification performance. First, in the step of feature engineering, to address the problem of that the original unstructured event logs are unsuitable for numerical calculation in deep CNN models, we propose a novel and effective log preprocessing method, which include building categories dictionary libraries, filtering abundant information, generating numerical semantic feature vectors by calculating and combining the semantic similarity values for filtered log events. Additionally, in the step of learning, we measure a series of deep CNN algorithms with varied hyper-parameter combinations by using standard evaluation metrics, and the results of our study reveals the advantages and potential capabilities of the proposed deep CNN models for log classification tasks on cluster systems. The optimal classification precision of our approach is 98.14%, which surpasses the popular traditional machine learning methods, and it can also be applied to other large-scale system logs with good accuracy. Just like the experiment results, different choices of learning algorithm do result in performance numbers varying, and subsequently careful feature engineering enables promoting performances, thus both of approaches contribute to best learning model finding.

***Index Terms*—Cluster logs; Event classification; Deep learning; Convolutional neural network; Numerical semantic feature**

I. INTRODUNCTION

With the widespread development of cloud computing, cluster systems are becoming increasingly complex, and the components in the entire system are also becoming diverse [30]. While the system logs describe the status of each component and record the system operational changes, such as the starting and stopping of services, software configuration modifications, software execution errors and hardware faults, and so on [10]. So analyzing these system logs is an universal and effective approach for automatic system management, monitoring, troubleshooting, failure prediction and root causes diagnosis [5][22][4]. Based on the above log-based analysis, if system administrators want to locate or predict which components/services have errors or failures, the general measures may be finding the event category[1] that log data belong. In addition, accurate log classification, is helpful for system administrators to constantly observe the health of the system and locate the root failure for anomaly diagnosis, task scheduling, and performance optimizations [11]. So log classification is an important issue in the field of log fault analysis and failure prediction, and it has been enjoying a growing amount of attention.

Currently, several traditional statistical methods and machine learning algorithms for automatic log classification have been developed, such as support vector machines (SVM) [**?**], density-based methods [2], clustering analysis-based methods [3], probability based approach [30], and model based approach [10], and so on. However, with the increase of log data volume, the traditional methods have certain deficiencies in classification accuracy and time efficiency. Surprisingly, deep learning approaches, are capable of handling complex scenarios of huge data, due to their more promising effectiveness in discriminative feature learning process. And it also can automatically implement dimension reduction to produce the optimal features set for categorization tasks.

In this paper, we develop a log classification system that is built on novel semantic-based numerical feature extraction and deep learning methods,which can predict the event category that the system log belongs to. Firstly, in order to address the problem of that the original unstructured event logs are unsuitable for numerical calculation in our classification models, a novel and effective log preprocessing method was proposed, which include building categories dictionary libraries, generating numerical semantic feature vectors by calculating and combining the semantic similarity values for filtered log events. Then we implement the log classification system based on deep CNN models, and set a series of experiments by adjusting various hyper-parameters configuration. The evaluation results show that our novel approach can classify log event automatically and effectively, and obtain a competitive precision 98.14%, which is outperforms the result of some other machine learning algorithms. In addition, the trained CNN model is applied to CMRI-Hadoop and bluegene/L logs and also obtains good accuracy for category prediction.

[1]The event category mentioned in this paper mainly refers to the location or service where the log event occurred.

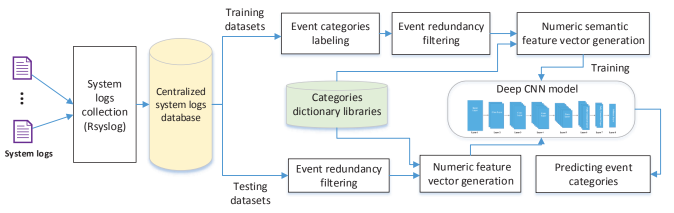

System logs
System logs collection (Rsyslog)
Centralized system logs database
Training datasets
Event categories labeling
Event redundancy filtering
Numeric semantic feature vector generation
Training
Categories dictionary libraries
Deep CNN model
Testing datasets
Event redundancy filtering
Numeric feature vector generation
Predicting event categories

Fig. 1. The working procedure of the log classification approach based on deep CNN.

II. Background and Goals

System logs record the information of hardware, software and system problems, and they are the primary resources for implementing dependability: they track system behaviors by accurately recording detailed data about a systems changing states [4], and users can use them to check the cause of system errors for log fault analysis; or they mine the signs of failures from system logs, and predict system failures, and so on.

Especially, we adopt an example to illustrate the procedure of log fault analysis: when an administrator analyzes the system logs, the first step is to determine whether the logs are part of the fault records. In this step, the most regular method is to find the error related keywords, such as, the administrator can determine the fault logs due to the keywords “fatal/failed/error” that included in the log events, or whose severity is “error/crit/alert/emerg/panic/failure/fatal”. The second step is log analysis. That is, the system administrator can check the details of the log information to determine the cause of the fault. For example, the first row in Table I shows *“Could not find keytab file: /etc/libvirt/krb5”* which indicates a filesystem problem. The second row shows *“No DHCPOFFERS received”* which indicates that system can not get the address dynamically through DHCP, which means the network is unavailable. Further more, the system administrators can consider that the first log event belong to the filesystem fault, and the second log event belong to the network fault. So the administrators can deal with the semantic analysis of the log events and determine the fault type in advance. Simultaneously, given the sets of common categories, the administrators also can categorize the messages reported by different components into the prescribed categories.

In this paper, we focus on the categories classification of log events, that is, based on the *msg* information of system logs, we design and implement a log classification system for classifying the log events into the specified event categories.

III. Log Classification System

In this section, we explain the proposed automatic log classification system based on deep CNN models, and the working procedure is shown in Fig. 1.

A. Collecting System Logs

The system logs are generated by machines, but they are not categorized by the systems, and it is too tough and time-consuming to identify by human. In this paper, the log collection tool Rsyslog [12] is used to collect the system logs from the the entire target nodes. We makes Rsyslog as a daemon and gathers the system logs from each node of cluster systems uninterrupted, and the format of collected logs includes: *timestamp, node, facility, severity* and *msg*. Here, the work in this paper is classifying the event categories based on *msg* information, and the examples of *msg* are listed in Table I. Then, the collected system logs are stored in the centralized system logs database, and divided as training datasets and testing datasests.

B. Event Categories Labeling

We can see from the collected log format in Table **??**, there is no fine-grained event category information. Before training the event categories classification model, we need to check and label all training datasets manually by assigning each of them a specified class C_k. By analyzing the collected system logs manually on the basis of the semantic meaning of the log message, we label the categories of log *msg* information into 13 event categories, which are shown in Table I.

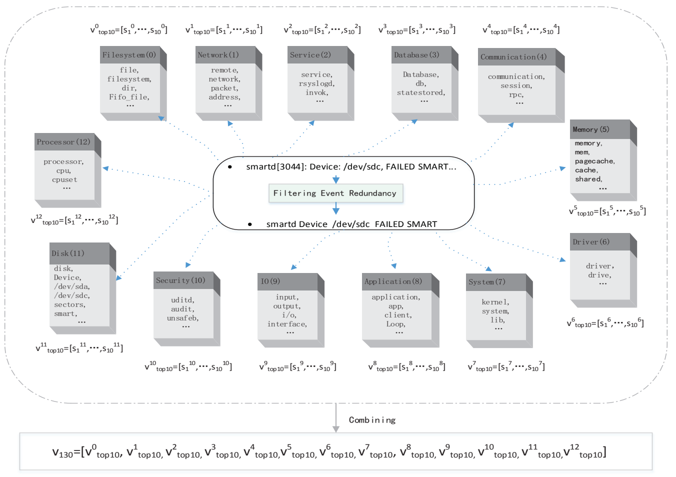
For example, based on the experience of administrators, if the main idea of the log message conveys some security information, then we assign this event as the Security category, and if the log message conveys some application fault, then we label this event as the Application category, so forth and so on. Therefore, the entire labeling task is done by human, and it is a tough and time-consuming job, which takes two months to complete the labeling work by two research staffs.

TABLE I
Categorization and Labeling of Log Events.

Example of msg	**Category**	**Label**
Could not find keytab file: /etc/libvirt/krb5	Filesystem	0
No DHCPOFFERS received	Network	1
Wake up new task 0xd3/0x120	Service	2
Error dropping database (cant rmdir testdb)	Database	3
WARNING: Unable to determine session	Communication	4
Read-error on swap-device (253:1.16968112)	Memory	5
Error: no connection driver available for qemu	Driver	6
Respawning too fast: disabled for 5 minutes	System	7
Application load error	Application	8
Buffer I/O error on device dev sda	I/O	9
Security real capable no audit	Security	10
FAILED SMART self-check. Back up now	Disk	11
Selected processor does not support 'strexb'	Processor	12

C. Building Categories Dictionary Libraries

Then, we have built up a dictionary libraries for each event category, which are used for converting log message text to numerical data. In this step, we choose the most frequent words by TF-IDF [26] method from labeled log messages, and make them as the keywords for building dictionaries for different log events. That is, the every category dictionary library contains tens of keywords, that represent the characteristics of different log events. Here, we define $KW_j^{C_k}$ as the j-th keyword in the event category C_k, and Fig. 2 give some examples of the keywords in each event category. In fact, the degree of



```
smartd[3044]: Device: /dev/sdc, FAILED SMART...
smartd Device /dev/sdc FAILED SMART
```

$v^{0}_{top10}=[s_1^{0},\cdots,s_{10}^{0}]$
$v^{1}_{top10}=[s_1^{1},\cdots,s_{10}^{1}]$
$v^{2}_{top10}=[s_1^{2},\cdots,s_{10}^{2}]$
$v^{3}_{top10}=[s_1^{3},\cdots,s_{10}^{3}]$
$v^{4}_{top10}=[s_1^{4},\cdots,s_{10}^{4}]$
Filesystem(0) file, filesystem, dir, Fifo_file, ...
Network(1) remote, network, packet, address, ...
Service(2) service, rsyslogd, invok, ...
Database(3) Database, db, statestored, ...
Communication(4) communication, session, rpc, ...
Processor(12) processor, cpu, cpuset ...
$v^{12}_{top10}=[s_1^{12},\cdots,s_{10}^{12}]$
Memory(5) memory, mem, pagecache, cache, shared, ...
$v^{5}_{top10}=[s_1^{5},\cdots,s_{10}^{5}]$
Filtering Event Redundancy
Disk(11) disk, Device, /dev/sda, /dev/sdc, sectors, smart, ...
$v^{11}_{top10}=[s_1^{11},\cdots,s_{10}^{11}]$
Driver(6) driver, drive, ...
$v^{6}_{top10}=[s_1^{6},\cdots,s_{10}^{6}]$
Security(10) uditd, audit, unsafeb, ...
IO(9) input, output, i/o, interface, ...
Application(8) application, app, client, Loop, ...
System(7) kernel, system, lib, ...
$v^{10}_{top10}=[s_1^{10},\cdots,s_{10}^{10}]$
$v^{9}_{top10}=[s_1^{9},\cdots,s_{10}^{9}]$
$v^{8}_{top10}=[s_1^{8},\cdots,s_{10}^{8}]$
$v^{7}_{top10}=[s_1^{7},\cdots,s_{10}^{7}]$
Combining
$v_{130}=[v^{0}_{top10}, v^{1}_{top10}, v^{2}_{top10}, v^{3}_{top10}, v^{4}_{top10}, v^{5}_{top10}, v^{6}_{top10}, v^{7}_{top10}, v^{8}_{top10}, v^{9}_{top10}, v^{10}_{top10}, v^{11}_{top10}, v^{12}_{top10}]$

Fig. 2. The example of building categories dictionary libraries, filtering event redundancy and generating the numeric semantic feature vector.

keywords overlap between different dictionaries across event categories is below 5%.

D. Event Redundancy Filtering

Because the irrelevant or redundant information normally brings much noise to feature extraction tasks, and may result in inferior classification performances, so removing the noise from the raw data is helpful for increasing the classification accuracy [15]. Here, we remove the noises by filtering out the event redundant information through stop-words and punctuations. And the stop-words reference Long Stop-word List[2] and punctuations comes from all commonly used symbols in the English language.

E. Numeric Semantic Feature Vector Generation

Since the filtered *msg* information is the unstructured log text, which is unable to be processed in the neural networks, we propose a novel and effective numeric semantic feature vector generation approach for log format conversion. The general idea is utilizing the numerical feature vector to represent the semantics of each *msg* information, which is helpful for efficient learning from the temporal events, and also can protect the sensitive information with respect to system operations.

1) Calculate Semantic Similarity: There are some ways to calculate semantic similarity, including but not limited to Euclidean Distance, Cosine Similarity and Pearson Correlation Coefficient. However, for a system log containing a large number of words, the above method will takes a lot of space and consumes a large number of computing resources. Therefore, in order to avoid these problems, we choose Levenshtein Distance [9] to measure semantic similarity. We define MSG_i as the msg information that can represent the log event. For every MSG_i, we calculate the semantic similarity value between them and every keyword in the event category, which according to the formula 1:

[2]http://www.ranks.nl/stopwords/

$$S(MSG_i, KW_j^{C_k}) = \frac{1}{lev(MSG_i, KW_j^{C_k})} \times 10 \qquad (1)$$

where $lev(MSG_i, KW_j^{C_k})$ refers to the Levenshtein Distance that measuring the difference between sequence MSG_i and sequence $KW_j^{C_k}$.

2) Combine feature vectors: For each MSG_i, after calculating the semantic similarity with each keyword that in event category, there are m semantic similarity values for each event category (here, m represents the number of keywords in a category dictionary library). Since the number of keywords may be large, the feature vector generated by combining the semantic similarity values may has large dimensions. More over, the large dimension of the feature vector may lead to time-consuming training for deep CNN models. So we merely select top n $(1 \leq n \leq m)$ according to the descending order of semantic similarity values, and then combine the semantic similarity values of all event categories into a feature vector. In our experiments, we just select top 10 $(n = 10)$ semantic similarity values for each event category and combine them. For example, for the event category 'Filesystem(0)', we select the top 10 semantic similarity values and generate a vector $V^0_{top10} = [S^0_1, ..., S^0_{10}]$, and by this way, we can generate the similar feature vectors such as V^1_{top10}, V^2_{top10}, and so on. At last, we can combine these feature vectors of all 13 event categories into a 130-dimensional numerical feature vector $V_{130} = [V^0_{top10}, V^1_{top10}, ..., V^{12}_{top10}]$.

In addition, the numeric features gained in this way not only retain certain textual structures of the original log events, but also maintain the differentiation among a variety of event types.

3) Normalizes: Generally, normalization is the final step before the log classification algorithm being executed. The min-max normalization which usually normalizes every feature into a [0, 1] interval, is employed in our experiments. Ultimately, normalized numerical semantic features and matching labels are utilized as dataset to evaluate our proposed models.

F. Classification model based on Deep CNN

Deep learning model is widely employed as it is very efficient in a large number of scenarios, especially for huge amount of high-dimensional datasets. In this section, we develop a set of deep CNN models [27], which contain several convolutional layers, pooling layers, fully connected layers and softmax classifier layers. In our CNN models, each hidden layer is responsible for training the unique set of features based on the output of the previous layer. The activation of each unit is computed via feature maps from the convolutional layer by using the following function:

$$a^{l+1} = \sigma(W^l a^l + b^l) \qquad (2)$$

where a^l denotes the activation for the units in layer l, W^l is the weight matrix in layer l, b^l represents the bias in the layer l, σ is the non-linear activation function. Additionally, the activation function in the fully connected layers are also

ReLUs, i.e. $f(x) = max(0, x)$. The output of the model after the softmax layer yields the probability distribution of label classes.

Meanwhile, in the backpropagation phase, the difference between the actual output and the ideal output is calculated, and the weight parameters are adjusted in reverse direction according to the minimization error method, thus the parameters such as weight, bias and other parameters are completed.

IV. EXPERIMENTS AND EVALUATIONS

In this section, we conduct an extensive set of experiments to evaluate the effectiveness of our log classification system by using some standard evaluation metrics.

A. Experiment Settings

We use Rsyslog tool to gather totally 100,000 log event records in our distributed cluster systems, which spanning a period of January 2017 to May 2017. And then we decide to take 80% of the original data as training dataset and the rest for testing in the experiments. Before model training, we analyzed the data distribution characteristics of the training dataset. We fined that the distribution of event types in training dataset is very skewed. For example, the proportion of Driver, Communication, Network logs are respectively 60.20%, 11.62%, 9.88%, while the proportion of Database, Security, Processor are just 0.01%, 0.13%, 0.02%. We consider that the system logs are related to the system's behaviors during that time, and it may lead to the incline of the event categories.

In addition, training the neural network usually takes much time and requires advanced hardware settings, so our training and classification tasks of the proposed models are run on GPUs, which leads to a significant decrease in training time. Our experiment environment configuration is listed as below:

CPU: Intel Xeon E5-2630, 2.4 GHz; GPU: Nvidia Tesla M40; RAM: 64GB; OS: Ubuntu 16.04.

B. Evaluation Metrics

To make a fair comparison between our log classification models to evaluate which model performs best, we evaluate the effectiveness of log classification models through the four metrics: *Precision*, *Recall* and *F1_score* [21].*Precision* means the exactness of classification, the higher the precision we have, the lower the rate of false alarms we get. *Recall* indicates to the completeness of classification, the higher the recall is, the lower the false negative rate we obtain. Nevertheless, only a precision or a separate recall is incapable of evaluating the effectiveness of an classification method. Therefore, we introduce the *F1_score* (the harmonic mean of *Precision* and *Recall*).

C. Hyper parameters adjustment Results in Deep CNN

In this section, we implement the deep CNN models in TensorFlow [1] framework. Because the hyper parameters are crucial for model initialization, and some unsuitable hyper parameters settings have an adverse effect on model final results, so we review and compare the sensitivity of varied hyper-parameters to the performance of a set of proposed models.

More often than not, more parameters generally facilitate robustness of the model. Meanwhile, massive parameters usage increases the dimensionality of the computation space, and might bring in undesirable effects like sparsity. Moreover, some redundant or irrelevant features are likely to diminish the performance of classification models, too. Here we aim at discovering the optimal hyper-parameter settings for deep CNN architecture. To assess the sensitivity of varying hyper-parameters to the performance of our classification models with respect to log events, we tune the following hyper parameters: 1) the learning rate; 2) the number of the convolutional filter; 3) the number of the convolutional and fully connected layer combinations (e.g., 2+2, 5+3, 7+3); and 4) the dropout probability value. For the learning rate, we pick a value from the set 0.0001, 0.001, 0.01, 0.1, and the possible values for the dropout probability are 0.25, 0.5, 0.75, 1.0. We varied the number of the convolutional filter from 16, 32, 64 to 128. The overall detailed combinations of diverse hyper parameters are presented in Table II.

In addition, to achieve the goal of training and testing efficiently, our dataset are split into mini-batches of a size of 1,000 during per iteration. The weights are initialized randomly, dropout operations are included in every model. The input of all our models consists of 130 vector features and output vector is comprised of 13 category labels. As a result, the dimension of input and output is 130 and 13, respectively. Furthermore, we train and test our models in 10 different random training/testing dataset combinations to decrease the influence of the bias in all evaluations.

1) Learning rate: Learning rate is one of the most important hyper-parameters, therefore it is very important to understand how to set it correctly for achieving good performance. If the learning rate is too high, you may overshoot the error minimum; if it is too low, your training will take forever. In our experiments, we evaluate the sensitivity of the learning rate with the weight values setting to 0.0001, 0.001, 0.01, 0.1, the number of the convolutional layer and fully connected layer are both 2 while the dropout is 0.75. Table IV-C1 depicts the trends of Precision, Recall and F1-score when the learning rate is increased. The average Precision, Recall, F1-score and Accuracy of proposed deep CNN models all steadily increase when the learning rate weight decreasing from 0.1 to 0.0001. We can get best Precision when learning rate equals to 0.0001, the Recall also obtains best value when we set learning rate to 0.0001. The Accuracy of all models are 100% except the model with the learning rate at 0.1.

2) Convolutional Filter Number: The number of the convolutional filter is also an important hyper-parameter affecting the deep CNN network performance. We test our deep CNN models with varied numbers of the convolutional filter from 16, 32, 64 to 128, given that all proposed models are composed of five convolution layers and two top fully connected hidden layers having the learning rate identical to 0.0001 with one additional softmax layer employed to generate posterior

TABLE II
THE TUNING COMBINATIONS OF HYPER PARAMETERS FOR DEEP CNN MODEL.

Number of Layer	Fully-Connect Size	Learning Rate	Dropout Value
(Conv + Full)	(Conv5+Full2)	(Conv2+Full2)	(Conv5+Full3)
{2+2, 2+3, 2+4, 3+2, 3+3, 3+4, 5+2, 5+3, 5+4, 7+2, 7+3, 7+4}	{16, 32, 64, 128}	{0.0001, 0.001, 0.01, 0.1}	{0.25, 0.5, 0.75,1.0}

*Conv: the convolutional layer; Full: the fully connected layer.

TABLE III
THE PRECISION, RECALL, F1-SCORE OF MODELS (CONV2+FULL2) WITH DIFFERENT LEARNING RATE ON TESTING DATA.

Learning Rate	Precision (%)	Recall (%)	F1-score (%)
0.0001	**97.23**	**97.07**	**96.99**
0.001	96.95	96.87	96.77
0.01	83.52	88.34	84.13
0.1	52.53	72.48	60.91

* Conv2+Full2: two-layer convolutional operation and two-layer fully connected operation.

probabilities. Still, we set the initial dropout value to 0.75. The classification results in Table IV-C2 shows that increasing the number of the convolutional filter tends to promote the performance of the models on the test dataset. However, when the number goes up to 128, the performance goes bad, even worse than the outcome of the model with 16 convolutional filters. No doubt, the best results are achieved when setting the number of the convolutional filter to 64, which will be implemented in all our next models.

TABLE IV
THE PRECISION, RECALL, F1-SCORE OF MODELS (CONV5+FULL2) WITH DIFFERENT NUMBERS OF THE CONVOLUTIONAL FILTER ON TESTING DATA.

No. of Conv Filter	Precision (%)	Recall (%)	F1-score (%)
16	97.77	97.71	97.63
32	97.83	97.90	97.83
64	**98.02**	**98.00**	**97.99**
128	97.47	97.56	97.50

* Conv5+Full2: five-layer convolutional operation and two-layer fully connected operation.

3) Convolutional and Fully Connected Layer: Deeper networks consist of deep stack of convolutional neural layers and fully connected layers, and generally are well suitable for the computation of huge dataset, which is one of the main reasons why we build deep CNN models. Of course, the obvious drawback of deeper networks architecture is that they require more training time. We evaluate the performance of deep CNN models when setting different number of the convolutional layers and fully connected layers, with 64 convolutional filters, learning rate equaling to 0.0001 and dropout identical to 0.75. Table V provides more insights about the specific classification performance of a series of proposed models. Of which, the combination with five convolutional layers and three fully connected layers offers a better precision. When keeping the fully connected layers constant, it improves the performance of proposed models if we increase the number of the convolutional layers from 2 to 5; but it obtains poor performance when the networks having too many layers (here denotes 7 layers), too.

TABLE V
THE PRECISION, RECALL AND F1-SCORE OF MODELS WITH DIFFERENT LAYER COMBINATIONS (CONV+FULL) ON TESTING DATA.

No. of Conv+Full	Precision(%)	Recall(%)	F1-score(%)
2+2	97.26	97.17	97.10
2+3	95.98	94.44	94.94
2+4	94.77	94.11	93.56
3+2	97.64	97.77	97.68
3+3	97.34	97.28	97.24
3+4	97.29	97.24	97.20
5+2	97.77	97.71	97.63
5+3	**97.95**	**97.89**	**97.86**
5+4	97.36	97.38	97.31
7+2	94.22	94.37	93.64
7+3	97.46	97.58	97.43
7+4	94.26	94.87	94.44

* Conv+Full: the convolutional layer and the fully connected layer.

4) Dropout: Dropout represents the probability of retaining a previous hidden unit in the network [16]. We assess the effects of varying the size of dropout to our models, which have 5 convolutional layers and 3 fully connected layers with learning rate at 0.0001, while the other hyper-parameters are held constant. Table IV-C4 demonstrates the Precision, Recall, F1-score and Accuracy obtained by models with different dropout values. It can be observed that the performance of models drops smoothly when $0.5 \leq$ dropout ≤ 1.0. This can be expected because having big dropout value means too many units will turn on during training phase. The model with dropout value identical to 0.5 gains almost perfect performance, which is better than any other models mentioned above. This indicates that, this parameter setting is the optimal hyper-parameter combination for proposed models to handle with the log classification.

TABLE VI
THE PRECISION, RECALL AND F1-SCORE OF MODELS WITH DIFFERENT DROPOUT VALUE ON TESTING DATA.

Dropout Value	Precision (%)	Recall (%)	F1-score (%)
0.25	93.27	93.18	92.13
0.5	**98.14**	**98.08**	**98.11**
0.75	97.95	97.89	97.86
1.0	93.27	92.73	91.62

* Conv5+Full3: five-layer convolutional operation and three-layer fully connected operation.

5) The Detailed CNN Model: In our CNN model, the first convolutional layer filters the $10X13$ input matrix with 32 kernels of size $3X4$ with a stride of 2, the valid padding size is 2, following by 32 kernels of 2X2 max pooling with a stride of 2 and a valid padding of 2. The second convolutional layer takes the output of the first convolutional layer as input and filters it with 64 kernels of size $3X4$, followed by the max

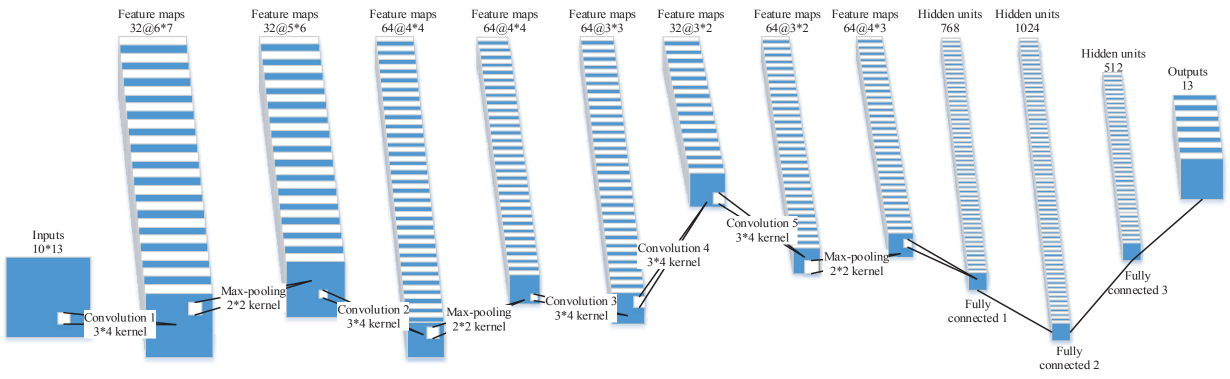


Fig. 3. The Detailed CNN Model.

pooling of size $2X2$. The third convolutional layer has 64 kernels of size $3X3$ connected to the outputs of the second convolutional layer. The fourth convolutional layer has 32 kernels of size $3X2$, and the fifth convolutional layer owns 64 kernels of size $3X2$ following by the valid max pooling of size $2X2$. The first fully-connected layers have 1024 neurons each with ReLU as activation function, and the dropout is 0.5. The second fully-connected layers have 512 neurons each, activation function is ReLU, the dropout still stays 0.5. The last fully-connected layers have 13 neurons each.

Meanwhile, the corresponding backpropagation output results for each layer are as follows: Conv5+Full3 corresponds to 8 weights respectively, and each weight corresponds to the same normal distribution matrix of the same kernel size with the mean=0, stddev=1. And the corresponding output neurons number is 32, 64, 64, 32, 64, 1024, 512, 13. Finally, we obtain the length of corresponding bias according to the neurons number, and then obtain the results of log classification based on the last bias length of 13.

D. Classification Results for Event Categories

In the best architecture of deep CNN model (Conv5+Full3, learning rate = 0.0001, dropout = 0.5) that we found, we give the classification results for the 13 event categories in the Table VII. We can conclude that, for most event categories, the classification effect of the best proposed model on testing data is excellent, particularly in the event categories of Service, Communication, Driver, Application and Security.

However, the classification effect is not good for predicting some event categories, such as, predicting the event categories of Filesystem, Database, I/O and Processor. So we also illustrate the heat map of normalized confusion matrix is presented in Fig. 4. From the heat map of confusion matrix, we see that quite a few existing category prediction errors are due to the confusions between Filesystem and Network, Database and Network, I/O and disk, Processor and System. One possible reason is that the data proportion of these log events are relatively small, thus recognizing them is pretty difficult, for example, the support number of Database and Processor event are just 2 and 10. The other possible reason is that some keywords of these categories dictionary libraries

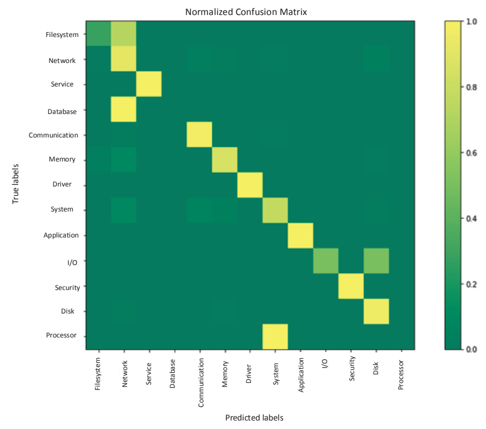


Fig. 4. Normalized confusion matrix heat map of the best deep CNN model (Conv5+Full3) on testing data.

are very similar, and leading to the classification errors. So our future works are providing more available data to reduce the unbalanced data distribution, and promote the robustness of the proposed model.

TABLE VII
THE PRECISION, RECALL AND F1-SCORE OF THE BEST MODEL (CONV5+FULL3) FOR EVENT CATEGORY CLASSIFICATION.

Categories	Precision	Recall	F1-score	Support
Filesystem	0.23	0.28	0.25	18
Network	0.84	0.91	0.87	911
Service	1.0	1.0	1.0	213
Database	0.0	0.0	0.0	2
Communication	0.98	0.98	0.98	2891
Memory	0.89	0.86	0.87	348
Driver	1.0	1.0	1.0	14,495
System	0.91	0.77	0.84	535
Application	1.0	1.0	1.0	132
I/O	1.0	0.50	0.67	4
Security	1.0	1.0	1.0	8
Disk	0.88	0.96	0.92	433
Processor	0.0	0.0	0.0	10
Avg/Total	0.98	0.98	0.98	20,000

* Conv5+Full3: 5 convolutional layers and 3 fully connected layers.

E. Comparison with Other Methods

In this section, to show the potentiality of our deep CNN models for complex classification issues, we test our preprocessed event logs on several popular machine learning models for log classification implementation. Specifically, based on the numeric semantic feature vectors of training dataset, we implement the following machine learning classification algorithms: AdaBoost [8], Decision tree [23], Multi-layer Perceptron(MLP) [13], Naive Bayes [25], Random Forest [6], Support Vector Machine (SVM) [20], Long short-term memory (LSTM) [24]. And we also optimized their parameters to get the best results.

Table VIII depicts the details of precision, recall and F1-socre results of different classification fashions. There is no doubt that our proposed deep CNN (5 conv + 3 fully) gains the highest precision (98.14%), followed by decision tree (96.92%) and random forest (95.59%), the next is MLP (94.99%), then followed by SVM with linear kernel (94.79%), LSTM (94.25%), Naive Bayes (87.48%), SVM with RBF kernel (86.15%) and AdaBoost (64.59%). The worst model is SVM with poly kernel, which only gets 56.92% precision.

TABLE VIII
RESULTS COMPARISON OF DIFFERENT MODELS FOR LOG CLASSIFICATION.

Method	Precision%	Recall%	F1-score%
AdaBoost	64.59	73.54	68.25
Decision Tree	96.92	96.90	96.62
MLP	94.99	94.87	94.43
Naive Bayes	87.48	89.07	87.60
Random Forest	95.59	94.37	94.30
SVM-Linear	94.79	94.25	93.18
SVM-RBF	86.15	89.47	87.45
SVM-Poly	56.92	73.70	63.47
LSTM	94.25	94.45	94.35
Deep CNN	**98.14**	**98.08**	**98.11**

F. Applying Model to Other Logs

Although our CNN model is built on a small dataset, the trained model can be applied to other large-scale system logs directly. For instance, we utilize the best model (Conv5+Full3) to predict event categories on CMRI-Hadoop and bluegene/L logs (these two logs have been labeled in our paper [5]), and the *Accuracy* [3] of category predictions reaches 96.29% and 95.6% separately. For these three logs, the iterations progresses of the best deep CNN model (Conv5+Full3) on training and testing data are shown in Fig. 5. From the experiment results we see that, because of the feature engineering on categories' keyword dictionary, the CNN model that we train on small log set can be applied to large-scale logs with good results.

V. Related Work

Various studies attempt to understand the meaning of logs, which may be useful for detecting faults or predicting failures in cloud computing systems [30]. And log classification is an effective way to understand the logs. For instance, s-ince the cloud computing systems contain different devices (e.g., processors, disks and drivers) with different software components (e.g., operating systems, middlewares, and user applications) [10], so automatic log event type classification can be useful for detecting the abnormal components or mining the root causes of abnormalities.

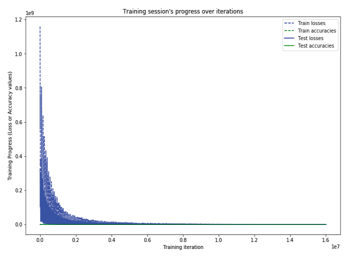
Currently, several techniques and algorithms for automatic log classification have been developed. Such as, Stearley [17] found that the fault type could not be detected from logs only by words, and the position of each word is a powerful indicator to distinguish different messages. Moreover, log classification can be viewed as a text categorization problem, whose goal is to assign predefined category labels to unlabeled documents based on the training set of labeled documents [14]. And Li et al. [10] tried to use the modified Naive Bayesian Model (NBM) and Hidden Markov Models (HMMs) to classify event logs based on the IBM Common Base Event (CBE) format. SLCT [18] and Loghound [19] are designed specifically to discover and classify rows of logs automatically.

Unlike the above studies, our study focuses on providing an effective deep learning method to improve the performance of log classification. Currently, it is widely accepted that deep learning algorithms are more well-suited for supervised classification problems, and achieve success in the fields of computer vision [27], audio recognition [28], etc., especially for dealing with large-scale data. In general, the more data a deep learning model is trained on, the higher accuracy it will gain. And the deep learning models can reduce the burden on selecting the features manually, owing to the capability of automatically grasping the relevant features. So in the field of log analysis, some works try to use the deep learning methods for log processing, fault detection and failure prediction, etc. For example, K. Zhang et al. [29] present a novel system that automatically parses streamed console logs and detects early warning signals for IT system failure prediction, which is the first work that employs LSTM (long short-term memory) for log-based system failure prediction. T. Islam et al. [7] propose a LSTM prediction method to identify application failures in cloud, whose goal is to predict the termination status (e.g., failed and finished etc.) of jobs or tasks from resource usage measurements or performance data. So we decide to use the deep CNN models for log categorization.

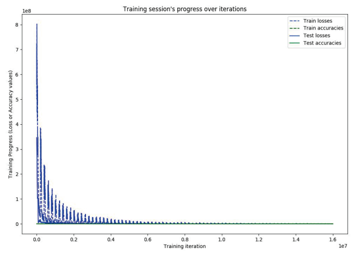
VI. Conclusion

In this paper, we propose a log classification system based on deep convolutional neural network for event category classification on distributed cluster systems. And the accuracy of log classification also shows the potentiality of deep CNN for classification tasks on distributed systems, and deep learning method is much better than other widely used machine learning methods. In other words, the conversion from unstructured texts into numerical values by using the Levenshtein distance can be regarded as *feature engineering*, and adjusting the hyper-parameters of deep CNN model can be regaraded as *learning algorithm*. Just like results mentioned

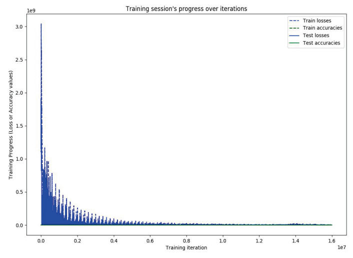
[3]Accuracy is defined as the ratio of correct classification number to the total number on a given test data set.



(a) Our collected log.



(b) CMRI-Hadoop log.



(c) Bluegene/L log.

Fig. 5. Iterations progress of the best deep CNN model (Conv5+Full3) on training and testing data.

in our paper, different choices of learning algorithm do result in performance numbers varying, and subsequently careful feature engineering enables promoting performances, thus both of approaches contribute to best learning model finding.

In short, the study has shown the remarkable classification results of proposed deep CNN approach in an efficient fashion. We believe this appealing deep learning approach is capable of providing high insights for understanding system logs without disclosing any business sensitive information. This area is still on-going research, and it also faces many challenges. The obvious but useful extension to this work would be to extend the experiments to a production environment, which is the direction of our efforts.

Acknowledgment

We are very grateful to anonymous reviewers. This work is supported by

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, and et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. In *arXiv preprint arXiv:1603*, 2016.

[2] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. Lof: Identifying density-based local outliers. In *ACM SIGMOD international conference on Management of data*, pages 93C–104, 2000.

[3] R. J. G. B. Campello, D. Moulavi, A. Zimek, and J. Sander. Hierarchical density estimates for data clustering, visualization, and outlier detection. *ACM Transactions on Knowledge Discovery from Data*, 10:1–51, 2015.

[4] X. Fu, R. Ren, S. A. McKeez, J. Zhan, and N. Sun. Digging deeper into cluster system logs for failure prediction and root cause diagnosis. In *IEEE International Conference on Cluster Computing(Cluster)*, pages 103–112, 2014.

[5] Xiaoyu Fu, Rui Ren, Jianfeng Zhan, Wei Zhou, Zhen Jia, and Gang Lu. Logmaster: Mining event correlations in logs of large-scale cluster systems. In *IEEE 31st Symposium on Reliable Distributed Systems (SRDS)*, 2012.

[6] T.K. Ho. Random decision forests. In *International Conference on Document Analysis and Recognition*, pages 278C–282, 1995.

[7] T. Islam and D. Manivannan. Predicting application failure in cloud: A machine learning approach. In *Proceedings of the 1st International Conference on Cognitive Computing*, 2017.

[8] B. Kgl. The return of adaboost. mh: multi-class hamming trees. *arXiv preprint arXiv:1312.6086*, 2013.

[9] V. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Journal of Soviet Physics Doklady*, 10:707–711, 1966.

[10] T. Li, F. Liang, S. Ma, and W. Peng. An integrated framework on mining logs files for computing system management. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 776–781, 2005.

[11] Y. Liang. Filtering failure logs for a bluegene/l prototype. In *International Conference on Dependable Systems and Networks(DSN)*, 2005.

[12] M. Peter. Centralised logging with rsyslog. *Canonical Technical White Paper*, 2009.

[13] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: the rprop algorithm. In *IEEE International Conference on Neural Networks*, pages 586–591, 1993.

[14] F. Sebastiani. Machine learning in automated text categorization. *ACM Computing Survey*, 34(1), 2002.

[15] M. R. Smith and T. Martinez. Improving classification accuracy by identifying and removing instances that should be misclassified. In *International Joint Conference on Neural Networks*, 2011.

[16] N. Srivastava, G. E Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Computers and Geotechnics*, 15(1):1929–1958, 2014.

[17] J. Stearley. Towards informatic analysis of syslogs. In *IEEE International Conference on Cluster Computing*, pages 309–318, 2004.

[18] R. Vaarandi. A data clustering algorithm for mining patterns from event logs. In *IEEE Workshop on IP Operations and Management(IPOM)*, pages 119–126, 2003.

[19] R. Vaarandi. A breadth-first algorithm for mining frequent patterns from event logs. In *In Proc. Intelligence in Communication Systems*, pages 293–308, 2004.

[20] A. Wang, Y. Zhao, Y. Hou, and Y. Li. A novel construction of svm compound kernel function. In *International Conference on Logistics Systems and Intelligent Management*, pages 1462–1465, 2010.

[21] C. Wang, V. Talwar, K. Schwan, and P. Ranganathan. Online detection of utility cloud anomalies using metric distributions. In *IEEE Network Operations and Management Symposium(NOMS)*, pages 96C–103, 2010.

[22] Y. Watanabe, H. Otsuka, M. Sonoda, S. Kikuchi, and Y. Matsumoto. Online failure prediction in cloud datacenters by real-time message pattern learning. In *International Conference on Cloud Computing Technology and Science*, pages 504–511, 2012.

[23] Wikipedia. Decision tree. https://en.wikipedia.org/wiki/Decision_tree, 2017.

[24] Wikipedia. Long short-term memory. https://en.wikipedia.org/wiki/Long_short-term_memory, 2018.

[25] Wikipedia. Naive bayes classifier. https://en.wikipedia.org/wiki/Naive_Bayes_classifier, 2018.

[26] Wikipedia. Tf-idf. https://en.wikipedia.org/wiki/Tf%E2%80%93idf, 2018.

[27] K. Zhang, J. Xu, M. R. Min, G. Jiang, K. Pelechrinis, and H. Zhang. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *In Proceedings of the 26th Annual International Conference on Machine Learning (ICML)*, pages 609C–616, 2009.

[28] K. Zhang, J. Xu, M. R. Min, G. Jiang, K. Pelechrinis, and H. Zhang. Unsupervised feature learning for audio classification using convolutional deep belief networks. In *Neural Information Processing Systems (NIPS)*, pages 1096C–1104, 2009.

[29] K. Zhang, J. Xu, M. R. Min, G. Jiang, K. Pelechrinis, and H. Zhang. Automated it system failure prediction: A deep learning approach. In *IEEE International Conference on Big Data (Big Data)*, 2016.

[30] D.Q Zou, H. Qin, and H. Jin. Uilog: Improving log-based fault diagnosis by log analysis. *JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY*, 31(5):1038C1052, 2016.