Detecting Anomaly in Big Data System Logs Using Convolutional Neural Network

Siyang Lu*, Xiang Wei*†, Yandong Li*, Liqiang Wang*
*Department of Computer Science, University of Central Florida, Orlando, FL, USA
†School of Software Engineering, Beijing Jiaotong University, China
Email: {siyang,yqweixiang,liyandong}@knights.ucf.edu, lwang@cs.ucf.edu

***Abstract*—Nowadays, big data systems are being widely adopted by many domains for offering effective data solutions, such as manufacturing, healthcare, education, and media. Big data systems produce tons of unstructured logs that contain buried valuable information. However, it is a daunting task to manually unearth the information and detect system anomalies. A few automatic methods have been developed, where the cutting-edge machine learning technique is one of the most promising ways.**

In this paper, we propose a novel approach for anomaly detection from big data system logs by leveraging Convolutional Neural Networks (CNN). Different from other existing statistical methods or traditional rule-based machine learning approaches, our CNN-based model can automatically learn event relationships in system logs and detect anomaly with high accuracy. Our deep neural network consists of `logkey2vec` embeddings, three 1D convolutional layers, dropout layer, and max-pooling. According to our experiment, our CNN-based approach has better accuracy (reaches to 99%) compared to other approaches using Long Short term memory (LSTM) and Multilayer Perceptron (MLP) on detecting anomaly in Hadoop Distributed File System (HDFS) logs.

***Index Terms*—CNN, Big Data, Log analysis, Anomaly detection.**

I. Introduction

Big data system plays an increasingly important role along with the rapid growth of massive data size. Several parallel computing frameworks have been widely used in real-world applications such as Dryad [1], Hadoop [2], and Spark [3]. When these big data systems process numerous data in parallel on distributed file systems [4], [5], [6], they also produce massive logs. In order to scrutinize problems in big data systems and improve their performance, these logs can be leveraged to mine crucial information for performance tuning and anomaly detection. However, analyzing these logs is very challenging. For example, Hadoop and Spark applications often demand long execution duration, thus a huge size of logs will be generated [7], [8]. Furthermore, each system may employ its own logging framework such as log4j [9] and self4j [10]; hence log formats could be diverse. Moreover, some unexpected events happening during the program execution might cause big performance degradation, or even failures [11], [12], [13]. Those scenarios are hard to be detected manually, even for system experts.

A log entry (log line) is considered as *anomaly* if it contains abnormal key words (*e.g.*, "error", "warning") or shows significant unexpected order in context, such as a Spark executor restarts repeatedly before it stops working. Classical anomaly detection has been studied for many years. Various algorithms and methods have been developed, such as basic key word searching, regulation expression matching, traditional statistical and machine learning approaches. It may incorrectly identify the anomalies and report false positives when searching anomalies with key words, or matching with regular expression. Hence, some techniques such as Support Vector Machine (SVM) and Principal Component Analysis (PCA) are often used to reduce the complexity of feature set to be analyzed and improve accuracy. However, the hidden relationships in extracted feature set are still very difficult to be analyzed by these aforementioned approaches, which often require more sophisticated approaches.

In recent years, deep learning approaches are leveraged in the log analysis domain to improve automation and accuracy. For instance, Long Short Term Memory (LSTM) and Recurrent Neural Network (RNN) are used by [14], [15] to detect anomalies with a high accuracy to avoid ad-hoc feature extraction. Within all deep learning methods, Convolutional Neural Network (CNNs) could be the most famous and widely used approach, which has obtained great achievements in computer vision. Due to the convolution layers, CNN-based approach can learn the hidden relationships with higher accuracy than other deep learning methods.

In this paper, we propose a CNN-based approach to explore the buried complex relationship in logs and detect anomalies effectively. First, we map the log keys to numbers and produce the embeddings using `logkey2vec`. Then, embeddings are fed into convolutional layers with different filters, where the width of filter is equal to length of a group of log lines. Next, a max-overtime pooling layer is applied to pick the maximum value for all features. Finally, we add a fully connected softmax layer to produce the probability distribution results. We compare the CNN-based approach with several other deep learning methods in anomaly detection for logs, and the CNN model shows the best performance.

The rest of the paper is organized as follows. Section II surveys the related work about anomaly detection for logs. Section III illustrates the methodology including log processing, CNN model design, and MLP model design for anomaly detection. Section IV evaluates our CNN approach. Section V compares our CNN approach with others. Section VI summarizes our methods and future work.

II. Related work

In this section, we survey existing log processing approaches and log-based anomaly detection methods.

A. Log Processing Approaches

Big data system logs are unstructured data printed in time sequence. Normally, each log entry (line) can be divided into two different parts: constant and variable. The constant part are the messages printed directly by statements in source code. Log keys can be extracted from these constant parts, where log keys are the common constant messages in all similar log entries. For example, as shown in Figure 1, the log key is "Starting task in stage TID partition bytes" in the log entry "Starting task 12.0 in stage 1.0 (TID 58, 10.190.128.101, partition 12, ANY, 5900 bytes)". The other part is the remaining after removing constant parts in log entries, which may contain variable keywords such as "12.0 1.0 58 10.190.128.101, 12, ANY, 5900".

```
17/02/22 21:04:02.259 INFO TaskSetManager: Starting task 12.0 in
stage 1.0 (TID 58, 10.190.128.101, partition 12, ANY, 5900 bytes)
.....
17/02/22 21:04:02.276 INFO TaskSetManager: Finished task 1.0 in
stage 1.0 (TID 47) in 14075 ms on 10.190.128.101 (1/384)
```

Fig. 1. Spark system log example

Detecting anomalies from system logs (such as Spark logs) requires log analysis, *i.e.*, analyzing root causes [16] using effective methods. Usually, log analysis consists of four main phases:

1) Parse unstructured raw logs into structure data by log parser techniques. There are two kinds of log parsing approaches [17]: heuristic and clustering. The clustering methods first conduct clustering based on distances result of logs, then create log template from each cluster. The heuristic methods count every word's appearance in these log entries and select frequently appeared words to be log events according to the predefined rules.
2) Extract log related features from parsed data. Different approaches may use different feature extraction methods (such as rule-based approach or execution path approach). There are several common window-based approaches for extracting different features such as session window, sliding window, and fixed window. Specifically, a session window is used for grouping log entries with the same session ID. A sliding window is used to slide forward in a certain step in the data and extract features with some overlaps. A fixed size of window can also be used to extract features.
3) Detect anomalies with extracted features, which is introduced in details in Section II-B.
4) Fix problems based on detected anomalies. There are many different ways to help fix problems based on detected anomalies, such as root causes analysis, anomalies visualization. For example, [14] leverages a decision tree to visualize the anomalies, and [16] uses a linear regression to compute the probability's of abnormal tasks.

B. Anomaly Detection Methods

We broadly classify the approaches of log-based anomaly detection into two categories: statistical approaches and machine learning approaches.

a) Statistical approaches: Statistical approaches do not need any training or learning phases, and mainly include rule-based methods, principal component analysis (PCA), and execution path extraction. Xu *et al.* [14] leverage abstract syntax tree (AST) to generate two log variable vectors by parsing system source code, then analyze extracted patterns from the vectors using PCA. Tan *et al.* [18] propose a general tool called SALSA, which uses state machine to simulate data flows and control flows in big data systems for anomaly detection in Hadoop's historical execution logs. Those methods have some limitations, for instance, specific rules and system knowledge are absolutely essential. Aguilera *et al.* [19] propose two statistical methods to discover causal paths (workflow) in distributed systems by analyzing historical logs and monitoring data from the traces of applications. Chen *et al.* [20] propose a statistical workflow execution tool named Pinpoint that leverages log traces to identify fault modules in J2EE applications. He *et al.* [17] evaluate total six supervised and unsupervised methods to provide guidelines for log-based anomaly detection. Safyallah *et al.* [21] detect system anomalies by mining common and frequent-sequence execution traces from system logs to detect anomalies. Fu *et al.* [22] leverage a rule-based approach to identify the log keys, and detect anomalies with log keys in distributed system log. In conclusion, statistical approaches can obtain a good accuracy for anomaly detection if rules and thresholds are set appropriately. Nevertheless, a special rule based approach is hard to adopt to deal with different system logs, hence statistical approaches are weak on portability.

b) Machine learning approaches: To avoid ad-hoc features in rule-based statistical approaches, machine learning techniques have been investigated for log-based anomaly detection. Support Vector Machine (SVM) is a classical supervised machine learning approach for classification. Liang *et al.* [23] build up three classifiers using RIPPER (a rule-based classifier), SVM, and a customized Nearest Neighbor method to predict failure events from logs. Moreover, Fulp *et al.* [24] use a sliding window to parse system logs and predict failures using SVM. Yadwadkar *et al.* [25] leverage a Hidden Markov Model (HMM), a learning approach, to detect anomalies. Lou *et al.* [26] propose a Bayesian-based learning approach to extract a construct graph from logs. However, those classical machine learning approaches are more time-consuming when handling large training sets.

Nowadays, deep learning becomes more and more popular in various fields, especial computer vision. Log analysis can also benefit from it. Long Short Term Memory (LSTM) network is a special Recurrent Neural Network (RNN) and

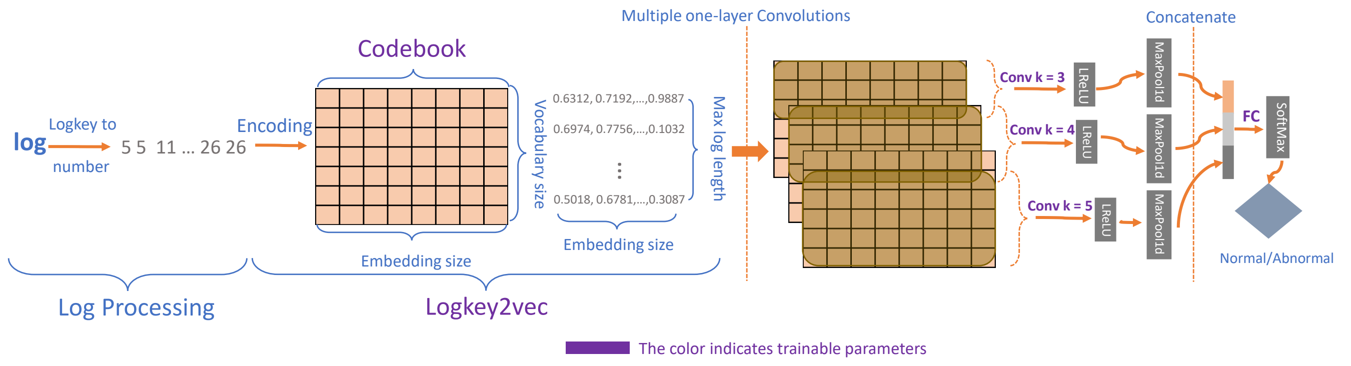

Fig. 2. Architecture of our CNN-based anomaly detection model.

is popular in the Natural Language Processing (NLP) domain. Min *et al.* [15] propose a tool called Deeplog by leveraging LSTM as its training model to detect anomalies within log execution path. Brown *et al.* [27] present an unsupervised RNN with attentions to discover relationships buried in system logs. In the NLP area, Kim *et al.* [28] propose a simple CNN model that can effective classify distributed word embeddings. CNN has been widely applied in computer vision areas, such as GoogLeNet [29], AlexNet [30] and other specific tasks [31], [32], [33], [34], [35]. More recently, deep Resnet [36] has achieved human-level recognition performance on image classification tasks. CNN is also used in text analysis [28]. Moreover, Jason *et al.* [37] show that the CNN model can be applied to discrete embeddings and achieve high accuracy. Now, various CNN models have been used in text classification.

In our prior work [16], we detect abnormal task and analyze root causes from Spark logs. It uses a parser to extract CPU related features, execution path features, and garbage collection features from raw logs. Then, a threshold is leveraged for detecting abnormal tasks with longer duration. After detecting abnormal in Spark log, we utilize a statistical rule based approach to create seven factors and calculate the weights of factors to decide the probability of root causes.

III. Methodology

In this section, we present our two-fold method. We first introduce our log processing, and then detail our CNN-based approach and MLP-based approach as the baseline.

A. Log Processing

The purpose of our log processing is to generate structural input for our CNN model. As the system log consists of multiple identifiers (defined by [14]), an identifier is an object and has a certain execution path. For example, `block_ID` is an identifier token in HDFS log, and `block_a` is an actual identifier and the execution path of `block_a` is a sequence that consists of three related log keys (*i.e.*, `Receiving block`, `PacketResponder for block terminating`, `Deleting block file`). Initially, a log template parser is used to find the frequent log constants, named log key. Then, we use another parser to analyze and filter the raw logs into structured data consisting of log keys (exclude useless information like timestamp of specific logs). Next, we encode each of the parsed log key with a unique number (*e.g.*, HDFS log has 29 log keys mapped to 29 numbers). Specifically, we count how many unique log keys in the whole data sets, and map each unique log key (parsed log entries) into a unique number. Finally, we leverage a session windows [17] to regroup those log keys to different sessions (group). After sorting those log keys (numbers) with execution order, we get a structured sessions. Thus, each session (group) includes one unique identifier and a series of related log keys (numbers), such as a session: `5 5 11 ... 26 26` in HDFS log belongs to one block (an identifier). Considering each vector represents an execution path which may vary based on environment settings (different orders), also each path may have different lengths. For example, some abnormal blocks in HDFS log will be killed after just being started, so this block only contains few log keys, which has a short length of vector. Hence, we pad 0 at the end of shorter vectors, and clip longer vectors to make each vector in the log files with the same length.

B. CNN-based Model

Neural network is a biologically-inspired approach for pattern recognition [38]. In regular fully connected networks, each neuron is fully connected to all neurons in the previous layer and Back-Propagation [39] is utilized to compute the error gradient [40]. However, it is not scaled well for high-dimension data such as images (*e.g.*, images are of size $32 \times 32 \times 3$ in CIFAR-10 [41]). Inspired by receptive fields of cat's visual cortex [42], Convolutional Neural Network (CNN) has been proposed to capture local semantic information instead of global information and defeat the overfitting issues in regular neural networks.

Basically, convolution is the core operation applied in the convolutional layers and it extracts features from local recep-

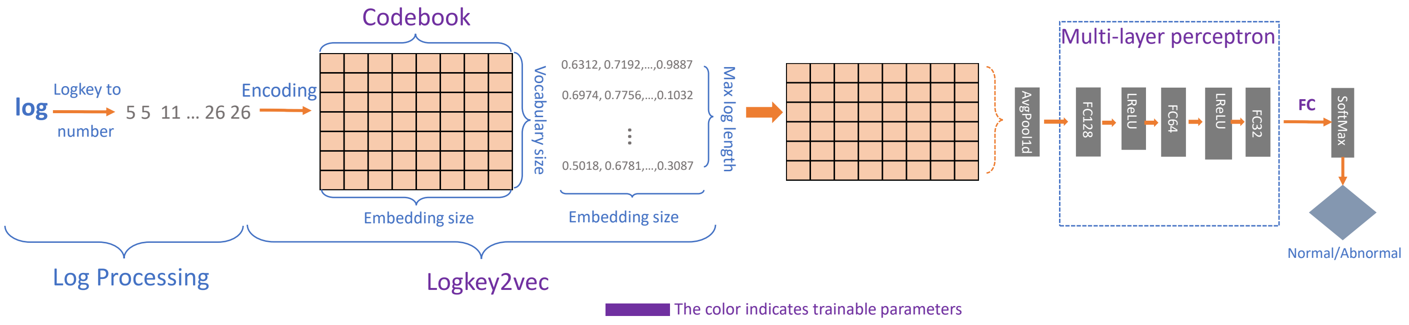

Fig. 3. Architecture of our MLP-based anomaly detection model.

TABLE I
NETWORK DETAILS AND SPECIFIC PARAMETERS IN OUR CNN MODEL

Layer	Output
Input: vectorized log, size: 1 x 50	--
Embedding with code book size: 29 x 128	Embedded log matrix size: 50 x 128 x1
Conv 1: [3,128,1,128], strides=[1, 1], padding="VALID" Leaky ReLU, max pool [1,48,1,1], strides=[1, 1]	48 x 1 x 128 1 x 1 x 128
Conv 2: [4,128,1,128], strides=[1, 1], padding="VALID" Leaky ReLU, max pool [1,47,1,1], strides=[1, 1]	47 x 1 x 128 1 x 1 x 128
Conv 3: [5,128,1,128], strides=[1, 1], padding="VALID" Leaky ReLU, max pool [1,46,1,1], strides=[1, 1]	46 x 1 x 128 1 x 1 x 128
Concatenate Conv 1, Conv 2, Conv 3, dropout 0.5	1 x 384
FC: [384,2]	2
softmax	

TABLE II
NETWORK DETAILS AND PARAMETERS IN OUR MLP MODEL

Layer	Output
Input: vectorized log, size: 1 x 50	--
Embedding with code book size: 29 x 128	Embedded log matrix size: 50 x 128 x1
Avg pool [1,50,1,1], strides=[1, 1], flatten	
Dropout	128
FC: [128*128], leaky ReLU	
Dropout	128
FC: [128*64], leaky ReLU	64
FC: [64*32], leaky ReLU	32
FC: [32*2]	2
softmax	

tive fields on feature maps of previous layer. An activation function (*e.g.*, Sigmoid, ReLU (Rectified Linear Units), Tanh) is performed as a non-linear transformation. Following [43], as shown in Eq. 1, the value of a unit at position (m, n) in the j^{th} feature map of the i^{th} layer can be denoted as $v_{ij}^{m,n}$:

$$v_{ij}^{mn} = \sigma\left(b_{ij} + \sum_{N}\sum_{p=0}^{P_{i-1}}\sum_{q=0}^{Q_{i-1}} w_{ij}^{pq} v_{(i-1)N}^{(x+p)(y+q)}\right) \qquad (1)$$

where b_{ij} denotes a bias function of this feature map, N indexes over the set of feature maps in the $(i-1)^{th}$ layer, P_i is the height of kernel and Q_i is the width of kernel, and w_{ij}^{pq} is the value of parameter.

As mentioned as before, Kim *et al.* [28] first propose a simple and effective CNN model based on `word2vec` [44] and vanilla CNN for sentence classification with static and non-static channels and get preeminent results in natural language processing.

Due to the fact that log file is also one special kind of text, log analysis can also benefit from the advances of NLP techniques. However, log analysis is different from the general NLP. The long span relationship widely exists in nature language context, such as a long sentence with complex structures. But logs only contain small amount of log keys. Moreover, the goal of anomaly detection is to look for unexpected execution path (log key sequences), which is a binary classification, whereas NLP tries to classify sentences into multiple categories.

As shown in Figure 2, in the embedding layer, we create a trainable matrix, *i.e.*, 29 $\times$ 128 codebook, to map each log key in a session into a vector. For example, in embedding process, the log key `5` in session group `5 5 11 ... 26 26` will be encoded to `0.6312, 0.7192, ... 0.9887`, and the whole session will be encoded as a matrix. We name this embedding process as `logkey2vec`. Different from word embedding that uses word as fine-grained unit such as `word2vec`, each log key will produce log embeddings based on the 29 $\times$ 128 codebook. The `logkey2vec` is a trainable layer optimized with gradient decent during the training of Neural Network. The codebook is used for mapping 1D vector to 2D matrix as CNN input, which is a more comprehensive mapping to enhance the relationships hidden behind logs.

The next part in CNN is convolutional layers, which convolute over the embedded log vectors with three one-layer convolutions (filters) in same time. According to our experimental study, we adopt three convolutional layers in parallel for CNN training after encoding layer, with size of 3 $\times$ 128, 4$\times$128, 5×128, respectively, as shown in Eq.1, where $P = 3, 4, 5$, and $Q = 128$. The activation function σ is Leaky Rectified Linear Unit (leaky ReLU or LReLU) shown in Eq. 2, due to that leaky ReLU can avoid over-fitting and solve the dead ReLU problem by setting first part of ReLU to non-zero (a small positive gradient). The dead ReLU problem means that some of neurons in the network may never be activated, hence, the parameters will never be updated. The causes of dead ReLU have two aspects. The first one is improper parameter initialization, and the second one is high learning rate setting which may lead to parameter updating too large. After three independent convolutional operations, a max-pooling layer is applied to concatenate output of the convolutional layers. While, for Leaky ReLU, as the gradient in all its domain will not be 0, it will feed back a informative update for each iteration. The max-pooling layer can also reduce over-fitting effectively by filtering out the weak related features, and leaving the strongest related features for next layer. Moreover, a dropout function is applied as a regularization in the second-to-last layer to prevent over-fitting. Finally, a softmax function is added in the output layer. The softmax function is shown in Eq. 3. Moreover, the parameter setting detail of CNN base model for each layer is shown in Table I.

$$\sigma(x) \begin{cases} x \quad if\ x \geq 0 \\ \\ 0.1x \quad if\ x < 0 \end{cases} \tag{2}$$

where x denotes the input before activator.

$$S_i = \frac{e^{a_i}}{\sum_{k=1}^{T}(e^{a_k})} \tag{3}$$

where a_i denotes the `i`th number of input, i = 1 to 2, T =2 in our implementation.

C. MLP-based Model

According to our empirical study, parameter tuning is very challenging for LSTM, it is difficult to train such a complicated model because of gradient vanish/exploding issues existing in Recurrent Neural Networks like LSTM. As a result, the accuracy for anomaly detection may decrease. Hence, we decide to use a simple and clear network with easy adjustable parameters as our baseline to compare with CNN in order to prove its efficacy. Therefore, we design a Multilayer Perceptron (MLP) as our baseline model and also train it on `logkey2vec` of HDFS logs. MLP is a kind of basic feed-forward artificial neural network. It consists of three components: input layer, hidden layers and output layer. Each hidden layer is activated with a non-linear function. BP is often utilized to update the weights of MLP. More than one hidden layers are designed to increase/decrease the complication of models. The output layer could be different depending on the objective function. The workflow of our MLP model for log based anomaly detection is shown in Figure 3. The input embedding stage is the same as CNN's `logkey2vec`, and it encodes vectors using the same codebook. The parameters of MLP model for each layer are shown in Table II, the hidden layers are three fully-connected layers without any convolutional layers, and the number of MLP hidden neurons for each fully connected (FC) layer is 128, 64, and 32, respectively. Following the CNN model, LReLU is also used as MLP's activation function. The output of FC layer is concatenated to a vector by an average-pooling layer.

IV. Evaluation

In this section, we first introduce the experiment setup, and then evaluate the accuracy of the CNN-based approach for detecting anomalies in HDFS data sets.

A. Experiment setup and data set

Our CNN-based approach is implemented in TensorFlow [45]. We compare the accuracy of our approach with other deep learning methods in log-based anomaly detection using HDFS log, a widely used benchmark dataset employed by other approaches [14], [15].

The HDFS log is a dataset generated from running over 200 days experiment in Amazon EC2. The data was first published by Xu *et al.* [14], and analyzed by many approaches such as SVM, PCA, logistic, and LSTM based anomaly detection. The raw log file is 1.55 GB and contains 11,197,954 log entries. Moreover, HDFS log records the states of each HDFS block during job execution time, and includes 29 unique log keys. Furthermore, the raw data is always parsed with session windows, and each line consists of unique blockId with related log keys in the parsed format. We leverage the parsed and labeled ground truth data, which is the same as [15]. It contains normal training set (4,855 parsed sessions), normal testing set (553,366 parsed sessions), abnormal training set (1,638 parsed sessions) and abnormal testing set (15,200 parsed sessions).

B. Accuracy of Our Methods

We evaluate CNN with our MLP baseline model and LSTM model on HDFS logs.

Due to the fact that both CNN and MLP are supervised approaches, and both require training before testing, we use normal training set from [15], and select 1% of abnormal testing data set as abnormal training set. Here, the remained 99% abnormal testing set combines with normal testing set as total testing set.

Those models are evaluated by the metrics listed blow: True positive (`TP`) represents the number of real anomalies that are correctly detected as anomalies by our approach. True negative (`TN`) represents the normal cases that are correctly identified as normal case. False positive (`FP`) presents the normal scenarios that are incorrectly identified as anomalies. False negative (`FN`) represents the abnormal log cases that are identified as normal. Based on the four metrics, we calculate the Precision (`P`), Recall, and F1-measure for each tested approaches. Precision is calculated by Eq. (4), which represents the correctly detected anomalies percentage in reported anomalies. Recall is calculated by Eq. (5), which shows the detected true anomalies in

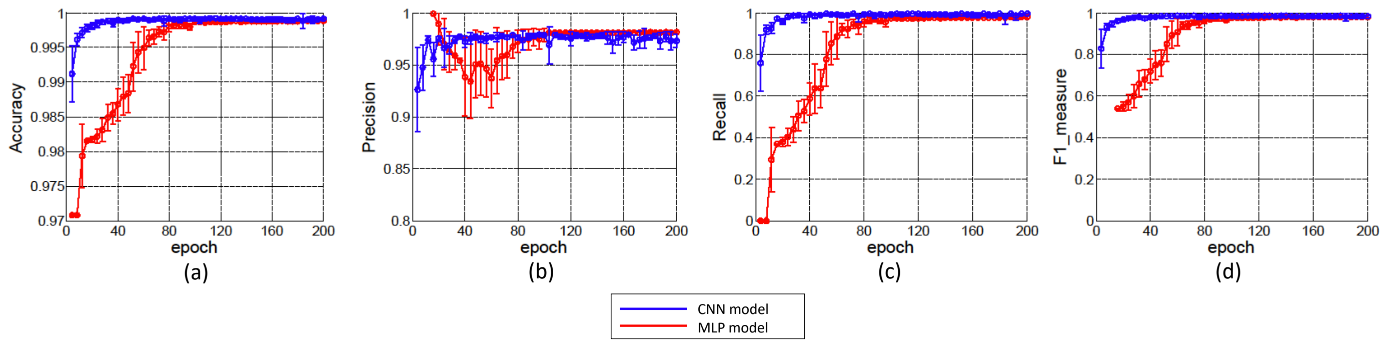

Fig. 4. Accuracy procedure of CNN-based approach and MLP-based approach on HDFS logs (a)Accuracy. (b) Precision. (c) Recall. (d) F1-measure

TABLE III
THE COMPARISON OF DIFFERENT MODELS ON HDFS LOG.

Model	Accuracy (%)	Precision	Recall	F1-measure
CNN	99.9±856e-05	97.7±068e-05	99.3±0035	98.5±0014
MLP	99.89±588e-05	98.12±918e-05	98.04±0036	98.08±0018
LSTM [15]	—	95	95	96

all real anomalies. F1-measure is calculated by Eq. (6), which represents the harmonic average of the P and recall.

$$P = \frac{TP}{(TP + FP)} \tag{4}$$

$$Recall = \frac{TP}{(TP + FN)} \tag{5}$$

$$F1\text{-}measure = \frac{2P{\cdot}Recall}{(P + Recall)} \tag{6}$$

C. Results

To compare the accuracy of our CNN model with LSTM and our MLP baseline model, we list all the evaluation metrics results in Table III. Our CNN achieves better results on all the metrics than the other two models.

Figure 4 compares the accuracy, precision, recall, and F1-measure of our CNN and MLP based approaches in each epoch of the training using HDFS logs. The red line is our CNN-based approach, and the blue line is the fully connected MLP-based approach without convolution layers. Figure 4 (a) shows that CNN has the higher accuracy. Although both CNN and MLP can achieve high accuracy finally, MLP starts with lower accuracy and slowly converges to high accuracy after 85 epochs. Figure 4 (b) presents that the precision curves of both models have some fluctuations; however, the curve of CNN model is much more stable than MLP. Moreover, CNN could converge in high precision after few epochs, and MLP converges after 100 epochs. Figure 4 (c) shows the recall of both models. The recall value of MLP starts at 0 and converges to 98.7 in 20 epochs, and CNN's recall is around 0.9 at the beginning, which is much higher than MLP's recall. Figure 4 (d) shows F1-measure of both models, where CNN could reach to a high accuracy in few epochs, but MLP converges till 90 epochs. All the evaluation metrics show that the MLP model converges slowly and is more time-consuming than the CNN model on the training of HDFS logs.

To evaluate if the embedding layer could impact the accuracy, we design an extra experiment by eliminating the embedding layer inside our MLP model. After MLP model trains with a series of log key vector directly without embedding process, we get the results with accuracy of 0.997, precision of 0.9732, recall of 0.95044 and F1-measure of 0.961726. Compared with the results shown in Table III, it demonstrates that the embedding process could cause big difference in the efficiency MLP for HDFS log classification. It is because that the embedding layer leverages codebook to encode vector into matrix, and this processing could learn comprehensive semantic representation of log.

V. DISCUSSION

In this section, we discuss potential reasons why the CNN model works better than the other two network approaches. Moreover, we analyze the significance of our CNN assembled the embedding layer.

A. CNN vs. MLP

There are two reasons that make the CNN and MLP models show different accuracy on HDFS log data. First, in a CNN network, the learning of weights not only takes the correlation between the horizontal embedding codes into account, but also the correlation between longitudinal entries in logs, which is a 2-dimensional convolution operation. More specifically, the method of MLP is relatively simple and fast, but the training procedure does not use the context information. Secondly, our CNN can mine more relationships in log context by leveraging multiple filters. After input embedding, each line of parsed data belongs to a unique identifier group, and consists of identifiers related log events in a sorted order. Namely, the parsed data presents an extracted execution path that has stronger co-correlation, where each identifier has a short and tight structure, and each log is positioned in a meaningful context.

For example, `Job start` is before `Job is running`, and `Job Finish` before `Job is detected`. Therefore, the CNN-based model can leverage convolution layer to extract those related features (relationships), which makes CNN get better performance.

B. CNN vs. LSTM

Long Short Term Memory networks (LSTMs) is an effective model for text classification, which shows more advantages than other approaches. Generally speaking, LSTM can store context information in each cell and continuously roll up the cell for next cell computation. Namely, the next cell's input is determined by the current cell's output and the prior cell's output. Hence, LSTM can mine long span information produced in a log execution path.

However, some factors may impact the LSTM's performance in log-based anomaly detection. First, the requirements of log analysis are quite different from NLP, where system logs have different structures from regular text files. Secondly, NLP requires word separation based on `word2vec`, and log analysis leverages log separation. Thirdly, as mentioned before, log has short sequence relationship among log entries. As each extracted execution path from log consists of many log entries (log keys), a log entries may not be related the log entries far from it. For example, in HDFS log, the start state of `Receiving block` of block and the finish state of `Deleting block file` are not very related. Thus, the long span information is not much useful for LSTM. Finally, the parameters of LSTM is hard to tune, which is very challenging to achieve a good performance.

When applying CNN to NLP, the word (log line) embedding can be naturally attached into the training of network, and then convert each log file vector into a 2-D matrix. Inside the matrix, each row indicates an encoding of word (log line), and column is the number of words inside the log file. In CNN, it naturally considers the content inside each log file, and is flexible to control its "memory length" by setting multiple convolutional kernel size, which turns out to be much easier to train compared with the recurrent-based network. For our implementation of log anomaly detection, there are only 29 words and more importantly, the log length is stable (average length is 19), thus, the encoding operation is much easier without losing much information. Hence, CNN is the preferred method for handling log data among the deep learning methods.

C. Logkey2vec embedding vs. Non-embedding

According to our experiment result, the network (CNN and MLP) assembled with embedding layer could achieve a better performance than our control group without embedding, and also the one that separately perform embedding layer before the DNN classifier, the result is presented in Section IV. There are two main reasons.

Firstly, to map a log key into a fixed length vector representation, we can catch a more comprehensive similarity measurement between two log keys, otherwise, the distance will be too straight forward. Such as the one hot embedding treats the each dimension of input data as a independent input, it marks difference for each kind of log key in the data set, but ignores the relationship hidden among each digit.

Secondly, even if the distance between keys can be calculated through word embedding, however, it still turns out to be difficult to set a proper library for this key-to-vector mapping, and finally positively affect the classification accuracy. For instance, the number of 5 and 25 may have a relatively large distance even by mapping them into vectors by a separate "word2vector" network, however, as the number 5 and 25 often appears next to each other in logs, it should be better to let the network learn their concurrency and finally set a proper mapping through a trainable embedding layer, and finally make the real classification-oriented network easier to be trained. By considering this, we naturally assemble the embedding layer into the classifier and finally get our "two stage one pass" DNN classifier.

VI. Conclusions and Future Work

This paper presents a novel Neural Network based approach to detect anomaly from system logs. A CNN-based approach is implemented with different filters for convoluing with embedded log vectors. The width of filter is equal to the length of a group of log entries. A max-overtime pooling is applied for picking up the maximum value. Moreover, multiple convolutions layers are employed for computing. Then, we add a fully connected softmax layer to produce the probability distribution results. We also implement a MLP-based model that consists of three hidden layers without any convolutional kernels. Our experimental results demonstrate that the CNN-based method can achieve a higher and faster detection accuracy than MLP and LSTM on big data system logs (HDFS logs). Moreover, our CNN model is a general method that can parse log directly and does not require any system or application specific knowledge.

For the further work, more complex system logs will be considered for training and testing. Furthermore, we plan to design an automatic log analyzer that can leverage deep learning approaches to detect anomalies and classify root causes into multiple classes.

VII. Acknowledgement

This work was supported in part by NSF-1622292.

References

[1] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," in *ACM SIGOPS operating systems review*, vol. 41, no. 3. ACM, 2007, pp. 59–72.
[2] "Apache Hadoop website," http://hadoop.apache.org/.
[3] "Apache Spark website," http://Spark.apache.org/.
[4] H. Zhang, H. Huang, and L. Wang, "Mrapid: An efficient short job optimizer on hadoop," in *Parallel and Distributed Processing Symposium (IPDPS), 2017 IEEE International*. IEEE, 2017, pp. 459–468.
[5] P. Guo, H. Huang, Q. Chen, L. Wang, E.-J. Lee, and P. Chen, "A model-driven partitioning and auto-tuning integrated framework for sparse matrix-vector multiplication on gpus," in *Proceedings of the 2011 TeraGrid Conference: Extreme Digital Discovery*. ACM, 2011, p. 2.

[6] L. Chen, W. Lu, E. Bao, L. Wang, W. Xing, and Y. Cai, "Naive bayes classifier based partitioner for mapreduce," *Ieice Transactions on Fundamentals of Electronics Communications Computer Sciences*, vol. 101, no. 5, pp. 778–786, 2018.

[7] V. Subramanian, L. Wang, E.-J. Lee, and P. Chen, "Rapid processing of synthetic seismograms using windows azure cloud," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*. IEEE, 2010, pp. 193–200.

[8] V. Subramanian, H. Ma, L. Wang, E.-J. Lee, and P. Chen, "Rapid 3d seismic source inversion using windows azure and amazon ec2," in *Services (SERVICES), 2011 IEEE World Congress on*. IEEE, 2011, pp. 602–606.

[9] C. Gülcü, *The complete log4j manual*. QOS. ch, 2003.

[10] "Self4j." https://www.slf4j.org/.

[11] H. Zhang, Z. Sun, Z. Liu, C. Xu, and L. Wang, "Dart: A geographic information system on hadoop," in *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*. IEEE, 2015, pp. 90–97.

[12] H. Zhang, L. Wang, and H. Huang, "Smarth: Enabling multi-pipeline data transfer in hdfs," in *Parallel Processing (ICPP), 2014 43rd International Conference on*. IEEE, 2014, pp. 30–39.

[13] H. Huang, J. M. Dennis, L. Wang, and P. Chen, "A scalable parallel lsqr algorithm for solving large-scale linear system for tomographic problems: a case study in seismic tomography," *Procedia Computer Science*, vol. 18, pp. 581–590, 2013.

[14] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *SOSP*. ACM, 2009.

[15] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 1285–1298.

[16] S. Lu, B. Rao, X. Wei, B. Tak, L. Wang, and L. Wang, "Log-based abnormal task detection and root cause analysis for spark," in *Web Services (ICWS), 2017 IEEE International Conference on*. IEEE, 2017, pp. 389–396.

[17] S. He, J. Zhu, P. He, and M. R. Lyu, "Experience report: system log analysis for anomaly detection," in *Software Reliability Engineering (ISSRE), 2016 IEEE 27th International Symposium on*. IEEE, 2016, pp. 207–218.

[18] J. Tan, X. Pan, S. Kavulya, R. Gandhi, and P. Narasimhan, "Salsa: Analyzing logs as state machines." *WASL*, vol. 8, pp. 6–6, 2008.

[19] M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthitacharoen, "Performance debugging for distributed systems of black boxes," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 74–89, 2003.

[20] M. Y. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer, "Pinpoint: Problem determination in large, dynamic internet services," in *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*. IEEE, 2002, pp. 595–604.

[21] H. Safyallah and K. Sartipi, "Dynamic analysis of software systems using execution pattern mining," in *Program Comprehension, 2006. ICPC 2006. 14th IEEE International Conference on*. IEEE, 2006, pp. 84–88.

[22] Q. Fu, J.-G. Lou, Y. Wang, and J. Li, "Execution anomaly detection in distributed systems through unstructured log analysis," in *Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on*. IEEE, 2009, pp. 149–158.

[23] Y. Liang, Y. Zhang, H. Xiong, and R. Sahoo, "Failure prediction in ibm bluegene/l event logs," in *Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on*. IEEE, 2007, pp. 583–588.

[24] E. W. Fulp, G. A. Fink, and J. N. Haack, "Predicting computer system failures using support vector machines." *WASL*, vol. 8, pp. 5–5, 2008.

[25] N. J. Yadwadkar, G. Ananthanarayanan, and R. Katz, "Wrangler: Predictable and faster jobs using fewer resources," in *Proceedings of the ACM Symposium on Cloud Computing*. ACM, 2014, pp. 1–14.

[26] J.-G. Lou, Q. Fu, Y. Wang, and J. Li, "Mining dependency in distributed systems through unstructured logs analysis," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 1, pp. 91–96, 2010.

[27] A. Brown, A. Tuor, B. Hutchinson, and N. Nichols, "Recurrent neural network attention mechanisms for interpretable system log anomaly detection," *arXiv preprint arXiv:1803.04967*, 2018.

[28] Y. Kim, "Convolutional neural networks for sentence classification," *arXiv preprint arXiv:1408.5882*, 2014.

[29] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich *et al.*, "Going deeper with convolutions." Cvpr, 2015.

[30] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[31] X. Wei, B. Gong, Z. Liu, W. Lu, and L. Wang, "Improving the improved training of wasserstein gans: A consistency term and its dual effect," in *International Conference on Learning Representation(ICLR)*, 2018.

[32] Y. Bian, C. Gan, X. Liu, F. Li, X. Long, Y. Li, H. Qi, J. Zhou, S. Wen, and Y. Lin, "Revisiting the effectiveness of off-the-shelf temporal modeling approaches for large-scale video classification," *arXiv preprint arXiv:1708.03805*, 2017.

[33] C. Gan, Y. Li, H. Li, C. Sun, and B. Gong, "Vqs: Linking segmentations to questions and answers for supervised attention in vqa and question-focused semantic segmentation," in *Proc. IEEE Int. Conf. Comp. Vis*, vol. 3, 2017.

[34] X. Long, C. Gan, G. de Melo, X. Liu, Y. Li, F. Li, and S. Wen, "Multimodal keyless attention fusion for video classification," in *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI 2018)*. AAAI Press, 2018.

[35] F. Li, C. Gan, X. Liu, Y. Bian, X. Long, Y. Li, Z. Li, J. Zhou, and S. Wen, "Temporal modeling approaches for large-scale youtube-8m video understanding," *arXiv preprint arXiv:1707.04555*, 2017.

[36] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[37] R. Johnson and T. Zhang, "Effective use of word order for text categorization with convolutional neural networks," *arXiv preprint arXiv:1412.1058*, 2014.

[38] C. Bishop, C. M. Bishop *et al.*, *Neural networks for pattern recognition*. Oxford university press, 1995.

[39] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," California Univ San Diego La Jolla Inst for Cognitive Science, Tech. Rep., 1985.

[40] Y. Ding, L. Wang, D. Fan, and B. Gong, "A semi-supervised two-stage approach to learning from noisy labels," in *IEEE Winter Conf. on Applications of Computer Vision*. IEEE, 2018.

[41] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

[42] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *The Journal of physiology*, vol. 160, no. 1, pp. 106–154, 1962.

[43] S. Ji, W. Xu, M. Yang, and K. Yu, "3d convolutional neural networks for human action recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 1, pp. 221–231, 2013.

[44] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[45] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning." in *OSDI*, vol. 16, 2016, pp. 265–283.