Log Anomaly Detection

Anomaly Detection on HDFS Logs Using CNN

Anqi Li
Irving K. Barber Faculty of Science
University of British Columbia
Kelowna, BC, Canada
anqi99@student.ubc.ca

Evelyn Sugihermanto
Irving K. Barber Faculty of Science
University of British Columbia
Kelowna, BC, Canada
evesugi@student.ubc.ca

ABSTRACT

System logs are messages that contain rich information about events that happen during system runtime operation. Detecting anomalies in large volume of system logs can often provide useful insights to the reason of a failure in a system.

As modern systems expand rapidly, the volume of logs being produced also increases. This leads to a problem as traditional approaches such as manually searching for a keyword through the logs and rule-based approaches becomes ineffective and inefficient.

In this project, we propose to build a system of log anomaly detection that can automatically go through the logs and accurately detect anomalies in a relatively short amount of time. The dataset that we will be using is the Hadoop Distributed File System (HDFS) 1 dataset that was obtained through Loghub, which is a database that stores large collection of system log datasets for research purposes [24].

Since the raw logs from the source are in an unstructured format, we first need to parse it to a more structured format so that it is compatible to be fit into the model. For the parsing part, we utilized the Drain and IPLoM (Iterative Partitioning Log Mining) algorithm that are available as part of the logparser package on github [22, 23]. After parsing the data, we then proceed to extract the features of the logs to produce a feature matrix that will serve as input for the next step. Convolutional Neural Network (CNN) is then used to detect whether a particular block is behaving normally or anomalous. As CNN is a supervised method and the raw logs are labelled, we can compute the accuracy of the model.

Using the abovementioned framework, we are able to automatically detect anomalies within the HDFS log file in approximately 2 hours with an accuracy of 99.9%.

CCS CONCEPTS

• Software and its engineering → Software maintenance tools;
• Computing methodologies → Machine learning approaches.

KEYWORDS

Log management, log dataset, log analysis, anomaly detection

1 Introduction

The Hadoop Distributed File System (HDFS) is a distributed file system that is used within the Hadoop framework. It is designed to run on commodity hardware, suitable for large datasets, and is highly fault-tolerant. While it is modern and advanced, it is not always flawless. Decrease in performance and failure sometimes occur during runtime. During runtime, HDFS produces system logs that records information about that particular runtime. For this project, we utilized the HDFS_1 log dataset available through Loghub [24]. These logs were created in a simulated environment using benchmark workloads on a Hadoop cluster running on 203 EC2 nodes. In total, this process yields 24 million lines of logs.

System logs often contain hidden valuable information that is sometimes overlooked. This is where anomaly detection comes in handy as it can help pinpoint the cause or the location of the decrease in performance or failure. As logs are rapidly increasing in volume, manually examining the logs through keyword searches becomes a tedious task and may take a relatively large amount of time. In the past, rule-based approaches have been developed in order to improve upon keyword searches. However, writing rules requires knowing which events are actually useful or relevant in detecting anomalies [18].

As we usually do not have this prior knowledge, we need an algorithm that can examine the relevancy of an event in the logs. TF-IDF (term frequency-inverse document frequency) is a widely used method in information retrieval that can assess how relevant a word is in a document. In this case, how important an event is in the logs. Together with a sliding window, we can extract relevant features of the logs for further analysis. To find the anomalies among the features, various algorithms have been deployed in the past such as Long Short Term Memory (LSTM) and Recurrent Neural Network (RNN) [5, 18]. However, Convolutional Neural Network (CNN) has gained popularity in recent years due to its high accuracy [25].

Hence, in this project, we will show an application of the combination of the methods above on the HDFS logs to automatically extract relevant features from the processed logs and predict anomalous behavior from the logs.

2 Background

With the rapid growth of technology, systems are evolving to a larger scale. To be able to store and process large-scale data, distributed systems such as Hadoop or Spark are commonly used. A few examples of businesses that used these technologies include banks, airlines, retailers, e-commerce, and many others. The users of these systems expect these systems to be stable and operate smoothly. Failure or reduction in performance will lead to interruption in the service and eventually result in a revenue loss for the business. Therefore, high availability and reliability is required for these systems to avoid system downtime. Additionally, it can also be costly to manually resolve the problem as it can take a lot of human resources and expertise of the system in the form of customer support.

According to a recent study, diagnosis of a problem in a system contributes 36-44% of the total cost of ownership related to support costs. Furthermore, downtime can add as much as 18-35% to the total cost of ownership [27]. Where the cost of ownership is defined as the cost to buy and operate a product or service. Not only it affects the business, but also the system vendor. More than 8% of the vendor's revenue and 15% employee costs are spent on customer support [28]. A good solution is then to automate the problem resolution where problems such as system failure or degradation in performance can be detected in a short amount of time. After the initial setup of the model, the anomaly detection algorithm is essentially easy and free for the company to reuse.

Though failure can come from both hardware and software, software failure usually takes longer to resolve. It was observed that the usage of log analysis can help accelerate the determination of the root-cause of the problem [26]. While the system is operating, logs are generated to record detailed information about its operation. These logs are the main data source to be used for analyzing problems or anomalies. However, the problem with analyzing logs is that they usually include other messages that do not relate to the cause of the problem. Different developers often have different templates for writing their logs as well making extracting features through pre-created template not generalizable. In response, automated log analysis that can extract features and predict anomalies automatically are highly sought after.

Through this project, we would like to present a start to finish model that can automatically parse raw system logs, extract features from relevant events in the logs, and differentiate anomalous behavior from normal ones. We will start with introducing a literature review of some of the available approaches and models in machine learning in Section 3. Then, we describe the details of the methodology that is used in this project in Section 4. Lastly, we will present the result of the methodology presented in Section 4 applied to the HDFS logs in Section 5.

3 Literature Review

Min Du et al. [5] proposed the DeepLog model, which is a Long Short-Term Memory (LSTM) deep learning network model combined with density clustering approaches. The model uses not only log keys but also metric values, which makes it able to capture different types of anomalies. Besides, they also successfully gave a way to make the model actively learn and update online with the new log data incoming in a streaming fashion so that it can incorporate and adapt to new log patterns. For the HDFS dataset, the model only needs a very small training set (less than 1%) but can achieve almost 100% detection accuracy on the test set.

Rakesh Bahadur Yadav et al. [6] summarized the main techniques used in research on log anomaly detection using Deep Neural Networks. For the feature extraction process, they used the Term frequency-inverse document frequency (TF-IDF) and the word2vec model including Continuous Bag Of Words(CBOW) and skip-gram model. Decision Tree, Support Vector Machine and PCA are widely used machine learning techniques. RNN, LSTM, Bidirectional LSTM and Autoencoder are most widely used Deep Learning models. Challenges in log analysis including the data are unstructured, instability, log burst, and availability of public dataset. It also provides a summary of log anomaly datasets.

Xiaoyun Li et al. [7] proposed the deep learning model SwissLog which worked with the Bi-LSTM model. They also proposed an advanced log parser based on a dictionary and without parameter turning. Bidirectional Encoder Representation from Transformers (BERT) is introduced to encode log templates, which makes the model robust to the format changing in log data. Besides, they also combined time and semantic embedding techniques together using a unified model.

Mengying Wang et al. [8] used multiple methods and compared their performance on log anomaly detection. First, they used the two common feature extraction algorithms TF-IDF and Word2vec respectively, and compared their performance. For the model, they used the LSTM model and compared it with other traditional machine learning models including Gradient Boosting Decision Tree (GBDT) and Naïve Bayes. The results show that that the LSTM's performance is better than other machine learning methods.

Vinayakumar R et al. [9] worked with stacked-LSTM (S-LSTM) which is formed by adding recurrent LSTM layer to the existing LSTM network in hidden layer. S-LSTM can learn long-range temporal dependencies with sparse representations without preliminary knowledge. The model achieved the highest accuracy of 0.996 with false positive rate 0.02 on the dataset provided by Cyber Security Data Mining Competition (CDMC2016).

Haixuan Guo et al. [10] proposed LogBERT, a self-supervised model based on Bidirectional Encoder Representations from Transformers (BERT), and used it in the two tasks: Masked Log Key Prediction (MLKP) and Volume of Hypersphere Minimization (VHM). They also compared it with other models including PCA, iForest, and LogCluster on 3 different datasets: HDFS, BGL, and Thunderbird. Experimental results show that LogBERT outperforms other models for log anomaly detection.

Hudan Studiawan et al. [11] implemented a Gated Recurrent Unit (GRU) Layer to detect the anomaly in OS logs, which is an improvement of LSTM as it only has the reset gate and the update gate. The model uses Softmax as Output Layer. Compared with other log anomaly detection techniques including Decision Tree, Logistic Regression, and SVM, the performance of logistic regression on all datasets are similar to the SVM.

Shayan Hashemi et al. [12] proposed the OneLog framework, which used a large character-based convolutional neural network (CNN) deep learning model. The model was evaluated on HDFS, BGL, Hadoop and Open-Stack datasets, respectively got F1 scores of 99.99, 99.99, 99.98, and 21.18. The biggest advantage of this model is that with the utilizing of character-level CNNs, it can successfully work with multiple datasets at the same time.

Xiaoyu Duan et al. [13] proposes a log anomaly detection model based on Q-learning algorithm. Q-learning algorithm is a kind of reinforcement learning model which stores all the information in a Q table and represents the Q-value between two states. The framework can not only detect multiple kinds of anomalies, but also provide the severity level of the events. The framework can also actively adapt to new log states and log patterns. From experiment, QLLog framework can perform log anomaly detection with 0.95 accuracy on both HDFS and OpenStack Datasets.

Yali Yuan et al. [14] proposes an Adaptive Deep Log Anomaly Detector, which is a deep neural network framework using LSTM network. The biggest advantage of this framework is that it can automatically update online and in real-time, which means it can adapt to new coming data and new patterns in the log data. The active learning approach can ensure the quality of the framework working with updated data patterns. The model can perform anomalies detection with F1 score of 0.95 on the Los Alamos National Laboratory (LANL) Cyber Security Dataset. The model is 97 times faster than other approaches.

Thorsten Wittkopp et al. [15] developed Attentive Augmented Log Anomaly Detection(A2Log) framework to deal with the current constrained assumptions of unsupervised log anomaly detection methods. It contains two stages. The first stage is the anomaly scoring process, which uses a self-attention neural network to get the Anomaly Score for each token sequence. The second stage is the anomaly decision process. They set the decision boundary based on the normal training dataset. The framework is evaluated on three real-world datasets from HPC systems: Blue Gene/L (BGL), Spirit, and Thunderbird (Tbird), and one industry dataset, and achieved good results.

Ermal Elbasani et al. [16] specifically applied the log anomaly detection framework to the health field on the Life-Log data. The Life-Log data are daily monitored health condition data collected through sensors. The framework used Recurrent Neural Network with Long Short-Term Memory(RNN-LSTM) model to help analyze the Life-Log data. Through actual evaluation, the proposed framework can achieve over 90% overall accuracy and outperforms any conventional machine learning techniques on accuracy.

Van-Hoang Le et al. [17] proposed NeuralLog, a log anomaly detection approach that doesn't need log parsing. The framework uses semantic vectors to represent the log data, and then used to detect anomalies through a Transformer-based classification model. The framework achieves an overall F1-score of 0.95 on the HDFS, BGL, Thunderbird and Spirit datasets.

4 Methodology

In this section we will present the details of the methodology used for the log anomaly detection framework. We will start with an explanation of the raw HDFS log data and then followed by a more comprehensive review of the methods used for log parsing, feature extraction, and the anomaly detection model.

4.1 Overview

Figure 1 shows an overview of the log anomaly detection framework that is used in this project. First, the raw log is parsed using the Drain and IPLoM methods so that we can see how different parsing methods affect the accuracy of the anomaly prediction. Second, the parsed log is then processed using the sliding window and TF-IDF technique in order to get a numerical feature matrix required for input to the next step. Third, the numerical feature matrix is then fit into a CNN model, more specifically the LeNet5 in order to distinguish normal and anomalous behavior.

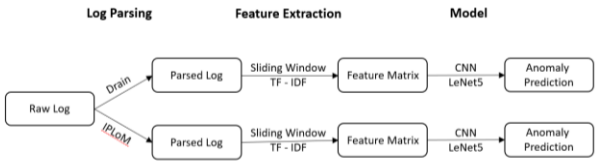

Figure 1: Log anomaly detection framework

4.2 Raw Log

Logs are routinely generated by systems as a way to maintain system states and runtime information. It usually comes in the form of a free-text. A line of log usually has constant and

variable parts. For the HDFS log, an example of a line of log would be in the form of "081109 203807 222 INFO dfs.DataNode$PacketResponder: PacketResponder 0 for block blk_-6952295868487656571 terminating". We can see that the log includes a timestamp at the beginning and message describing what happened at a particular time. In particular, "PacketResponder" and "for block" is an example of the constant part and the "0", "blk_-6952295868487656571" and "terminating" is the variable part. Figure 2 presents how the raw log data looks like.

```
081109 203807 222 INFO dfs.DataNode$PacketResponder: PacketResponder 0 for
block blk_-6952295868487656571 terminating
081109 204005 35 INFO dfs.FSNamesystem: BLOCK* NameSystem.addStoredBlock:
blockMap updated: 10.251.73.220:50010 is added to blk_7128370237687728475
size 67108864
081109 204015 308 INFO dfs.DataNode$PacketResponder: PacketResponder 2 for
block blk_8229193803249955061 terminating
081109 204106 329 INFO dfs.DataNode$PacketResponder: PacketResponder 2 for
block blk_-6670958622368987959 terminating
081109 204132 26 INFO dfs.FSNamesystem: BLOCK* NameSystem.addStoredBlock:
blockMap updated: 10.251.43.115:50010 is added to blk_3050920587428079149
size 67108864
081109 204324 34 INFO dfs.FSNamesystem: BLOCK* NameSystem.addStoredBlock:
blockMap updated: 10.251.203.80:50010 is added to blk_7888946331804732825
size 67108864
```

Figure 2: Sample of raw HDFS log

4.3 Log Parsing

Log parsing is necessary in order to transform the raw logs into something that we can work with. The log parsing method will create event templates for the logs such that the content of the template can be extracted. In this project, we will be using the Drain and IPLoM (Iterative Partitioning Log Mining) methods to parse the log into a more structured format. Implementation of both the Drain and IPLoM methods are available from the open-source logparser package on github. We choose one online (Drain) and offline (IPLoM) methods that have been tested to work well with the HDFS log [22, 23] to compare how the chosen log parsing method affects the accuracy of the results.

4.3.1 Drain

Drain [19] is a fixed depth tree based online log parsing method. It preprocesses incoming raw log messages by using regular expressions. The method allows user to incorporate domain knowledge through the regular expression specified. Then it searches for a log group by using a specially designed rule, guided by a parse tree with fixed depth to make the process faster. Upon finding a log group suitable for a message, the algorithm will match the log message with the log event stored in that log group. If none are found, the algorithm will create a new log group based on the log message. Drain selects the most suitable log group by calculating the similarity between the log message and the log event for each of the log group.

4.3.1 IPLoM

IPLoM (Iterative Partitioning Log Mining) [20] is an iterative, heuristic-based method for log parsing where first each word occurrences are counted, words that appear frequently are selected to be candidates for log events [29]. The implementation of the IPLoM method in the logparser package are described in the following four steps [30].

Step 1: First, the algorithm will create clusters and partition the logs based on its length.

Step 2: Now, each cluster of logs consist of logs of the same length. Each of the cluster are in the form of an $m \; x \; n$ matrix where m is the number of logs in each cluster and n is the length of the logs in each cluster. Next, the logs are partitioned by token position. Assuming that the column with the fewest unique words is the one that contains constants, call it the split word position, the algorithm partitions the cluster based on the split word position.

Step 3: In this step, the algorithm partitions the logs further according to their mapping relation by choosing two columns. The two columns that has the most frequently appearing words based on its unique word count are selected. There are four cases to the mapping relations namely, 1-1, 1-M, M-1, M-M. For the 1-1 relation, logs that have the same 1-1 relation are partition to the same column. For the 1-M and M-1 relations, if the M side of the columns contains constants, then the M side of the column is used to partition the logs in both cases. Else, the 1 side of the column is used to partition the logs. Lastly, the M-M relation are partitioned into one cluster.

Step 4: The last step is to extract the log template. The algorithm goes through all the clusters and create a log template for each cluster. The number of unique words in a column is counted for each column in a cluster. If a column only has one unique word, then the word is considered a constant. Else, the algorithm will consider the words as variables and replace them by a wildcard in the output.

4.4 Feature Extraction

In order to extract meaningful and relevant insights from the parsed log, we need to deploy a feature extraction method. Since it would be easier for the model to understand numerical values, we want to create a numerical matrix feature through the feature extraction process. We used the TF-IDF (term frequency-inverse document frequency) method along with a sliding window to do the feature extraction in this project.

4.4.1 Sliding Window

Before we can start to extract the features, we must first divide the log into groups of event sequence. The use of window is to partition the log into finite chunks [29]. Sliding windows have two components which are the window size and step size. The step size determines how far the window slides through the event sequence. The window size determines the size of the frame that captures the events in one step. The step size is normally smaller than the window size to allow for overlapping of windows in the event sequence.

4.4.2 TF-IDF

TF-IDF is a statistical method that is commonly used in information retrieval and text mining to assess the importance of a word to a collection of corpus or documents. In this case, the word will be the log events and the documents are the event sequence in each block id. TF-IDF is calculated by taking a product of its two parts which are the TF and IDF.

$$tfidf(t,d,D) = tf(t,d) \cdot idf(t,D)$$

The value of TF-IDF is proportional to the number of events offset by how frequently the event shows up in the event sequence. In other words, it takes into account an adjustment for common events that shows up in the event sequence.

Term Frequency, $tf(t,d)$,is defined as the frequency of term t within a document d divided by the sum of the frequency of term t in all documents [31]. In other words, TF is the relative frequency of a term in a document. The formula to calculate TF is shown in the equation below.

$$tf(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$$

Where $f_{t,d}$ is the number of terms t in a document d. The value of TF depends on how common a word is in a document and how long the document itself. Computing TF alone is not sufficient as common events will get a high score event though it may not be relevant to detect anomaly. That is why we also need to compute the inverse document frequency as an adjustment to ensure that only relevant events get a high value and irrelevant events will have a less prominent value.

Inverse document frequency, $idf(t,D)$, is the log inverse of the number of occurrences of term t in all documents D. It represents the amount of information a term provides, whether it is more common or rarer [31]. The formula to calculate the inverse document frequency is given in the equation below.

$$idf(t,D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

Where N is the number of documents and $|\{d \in D : t \in d\}|$ is the number of documents that contains the term t.

4.5 Supervised Model

Supervised learning models require labelled dataset to train. The HDFS log obtained from Loghub have been carefully labelled with normal or anomaly labels which enables us to deploy a supervised model to do the anomaly detection. We can split our data into a train and test set. The train set is used to train the supervised model. Once the model has generated its prediction for the labels, we can analyze the accuracy of the models by comparing the test set with the original labels.

The supervised model that we will be using is called the Convolutional Neural Network (CNN). Although it is more commonly used for analyzing images, we will show through the results in Section 5 that CNN works very well in predicting anomaly for logs. CNN takes a grid-like input such as pixels of an image. That is why we need to transform the parsed log data into something that the CNN can work with, that is, the numerical feature matrix. The main architecture of CNN commonly consists of convolutional layer, pooling layer, and fully connected layer.

The convolutional layer performs a convolution operation on the input. To explain what a convolution operation is, first, we must specify a filter to be used. A filter can be thought as a relatively small size matrix with a pre-specified number of rows and columns. This filter will slide through the input matrix, according to the sliding size called stride, performs a dot operation, and stores the result. The filter will then slide to the next position, repeat the operation until all the components of the matrix are passed through This operation is referred to as convolving. The output of this layer is called a feature map or activation map. Figure 3 illustrate how the convolutional layer works.

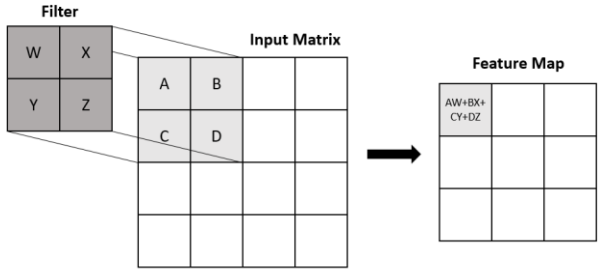

Figure 3: Illustration of the convolution layer

The pooling layer pools a specified tile of data by using a summary statistic in order to reduce the dimension of the feature map. The most common used tile size is $2x2$. Whereas the most common summary statistic used to pool the data are the maximum and average. Say the components of a $2x2$ tile of a feature map is [1,2,3,4], then max pooling will give 4 and average pooling will give 2.5 as a result. Figure 4 illustrate how the pooling layer works.

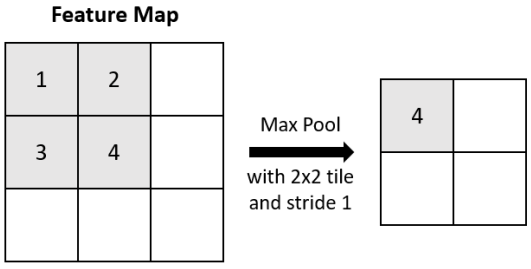

Figure 4: Illustration of the pooling layer

The fully connected layers connect neurons in the preceding and succeeding layers. It is simply a feed forward neural network. This layer will take a flattened input from the previous layer, perform a calculation of the matrix multiplication and addition of bias effect, then pass it through an activation function.

Apart from the three layers discussed above, non-linear layers are usually placed after the convolutional layer to incorporate non-linearity to the feature map. Several examples of non-linear layers include Sigmoid, Tanh, and ReLU.

The final adopted CNN model for this project is further discussed in section 5.1 of the experimental setup.

5 Experiments and Results

5.1 Experimental Setup

Dataset. We evaluate the proposed model on the HDFS [18] log dataset. The size of the dataset is over 16GB with normal/abnormal labels. The log set was collected by aggregating HDFS system logs, which contain one name node and 32 data nodes.

First, we split the dataset into a training set and a test set. We adopt 80% of log sequences for training and 20% of log sequences as the test set. Table 1 shows the statistics of the dataset.

Table 1: Statistics of evaluation dataset

Dataset	#Log Messages	#Blocks	#Normal Blocks	#Anomalous Blocks	% of Anomalous
Training	8940503	394503	382184	12319	3.12
Test	2235127	120537	118554	1983	1.65
Total	11175630	515040	500738	14302	2.78

Parse. We adopt Drain [19] and IPLoM (Iterative Partitioning Log Mining) [20] from the open-source tool Logparser to parse the log messages into log keys and to compare their performance on the final detection accuracy.

Drain used a fixed depth parse tree which encodes specially designed rules for parsing to accelerate the parsing process. IPLoM contains 4 steps: partition by event size, partition by token position, partition by search for mapping, and log template extraction.

After parsing, the unstructured raw log data was converted to log event templates (column EventTemplate) and structured log variables (column ParameterList). Table 2 shows an example from the parsed data.

Table 2: Example of parsed dataset

Content	EventTemplate	ParameterList
'Receiving block blk_-1608999687919862906 src: /10.250.19.102:54106 dest: /10.250.19.102:50010'	'Receiving block <*> src: <*> dest: <*>'	"['blk_-1608999687919862906', '/10.250.19.102:54106', '/10.250.19.102:50010']"

Feature Extraction. We first used the Term Frequency-Inverse Document Frequency (TF-IDF) model to measure the event count. The TF-IDF is an extensively used approach for feature extraction, which can provide a metric that can reflect how important each event is. After applying the TF-IDF function, each block is converted to a resulting TF-IDF values vector.

After that, we used sliding window event counts to convert the event sequence for each block id to a matrix. Figure 5 show an example of how the matrix for each block id is generated using window size 10. The actual window size we used on the HDFS dataset is 16.

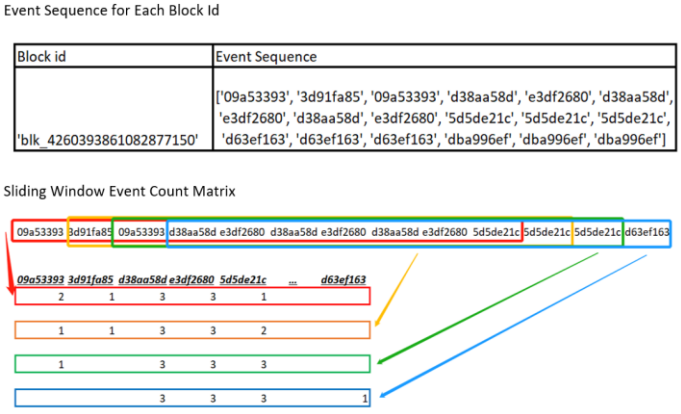

Figure 5: An example of sliding window event count matrix generation process

The final matrix is generated by multiplying the corresponding TF-IDF vector and the sliding window event count matrix for each block id.

Model. We adopted a CNN model which is modified based on the classic LeNet-5 convolutional network [21] provided in lab3. In total, the model contains 6 layers. 2 out of these 6 layers are convolutional layers (C1, C3), which are connected by two

maximum pooling layers (S2 & S4). The penultimate layer is a fully connected layer (F6), which is followed by the final output layer. The details are summarized below:

- The two convolutional layers are with 2x2 kernels, and are 16 and 32 filters respectively.
- The two max pooling layers are with 2x2 kernels.
- There are two MLP hidden layers with 120 and 84 nodes respectively.
- The whole network uses tanh sigmoid as activation function.
- The output layer uses softmax function with output labels representing normal and anomalous.

5.2 Experimental Results

Figure 6,7,8,9 show the cost of each batch and the model's accuracy of each epoch during the training process. We can see that the cost generally decreases to below 0.1, and the accuracy is generally above 0.999 for both the training and validation set.

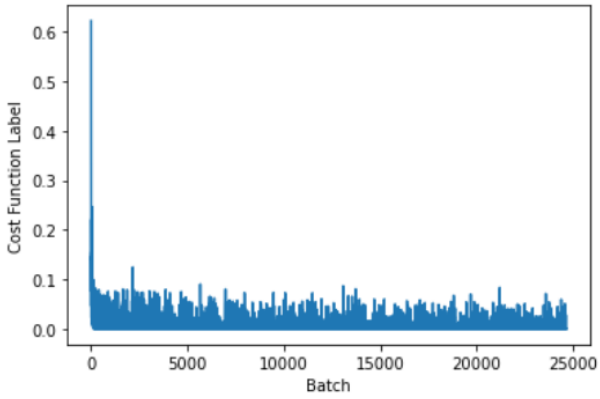

Figure 6: Cost of each batch with data parsed by Drain

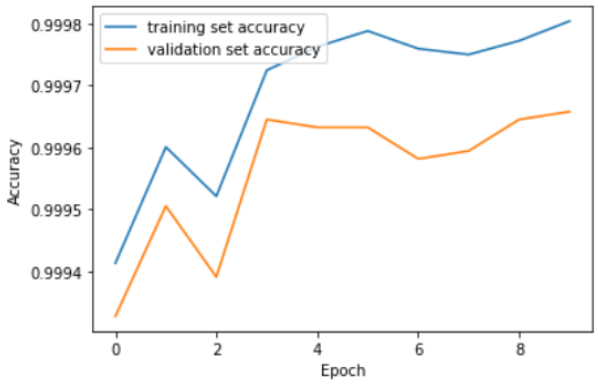

Figure 7: Accuracy on training and validation set with data parsed by Drain

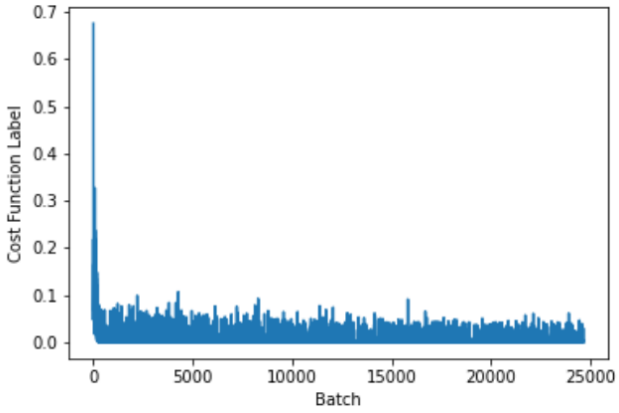

Figure 8: Cost of each batch with data parsed by IPLoM

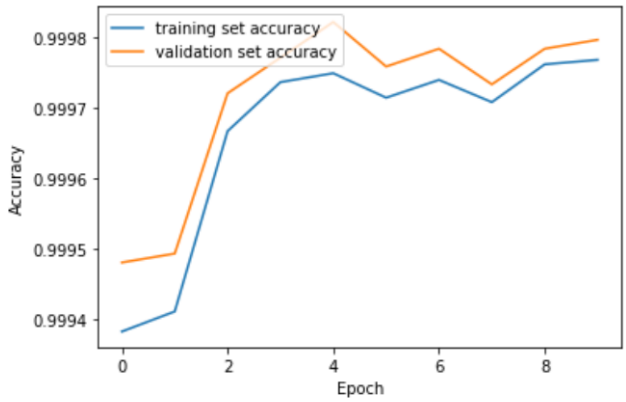

Figure 9: Accuracy on training and validation set with data parsed by IPLoM

Table 3 and Table 4 respectively show the confusion matrix and the performance metrics of the model on the training and testing set, parsed by the Drain/IPLoM model respectively.

Table 3: Confusion Matrix of the CNN Model using Drain/IPLoM parsing

Drain	Train	True Normal	True Anomaly
	Predicted Normal	305677	18
	Predicted Anomaly	44	9863
	Test	True Normal	True Anomaly
	Predicted Normal	118554	12
	Predicted Anomaly	0	1971
IPLoM	Train	True Normal	True Anomaly

	Predicted Normal	305728	27
	Predicted Anomaly		9801
	Test	True Normal	True Anomaly
	Predicted Normal	118554	9
	Predicted Anomaly	0	1974

Table 4: Performance Metrics of the CNN Model using Drain/IPLoM Parsing

Parsing model	Dataset	Accuracy	Precision	Recall
Drain	Train	0.999803	0.995558	0.998178
	Test	0.999900	1.0	0.993949
IPLoM	Train	0.999769	0.995329	0.997253
	Test	0.999925	1.0	0.995461

In conclusion, the experimental results with the CNN model have shown a very high performance with accuracy, precision, and recall over 0.99 both on the training set and testing set. However, for the two different parsing tools Drain and IPLoM, there's no big difference in the model's performance.

It was demonstrated in this study that the proposed deep learning model provided a better way to detect anomalies in log data than the conventional machine learning techniques with better accuracy.

REFERENCES

[1] Patricia S. Abril and Robert Plant, 2007. The patent holder's dilemma: Buy, sell, or troll? *Commun. ACM* 50, 1 (Jan, 2007), 36-44. DOI: https://doi.org/10.1145/1188913.1188915.

[2] Sten Andler. 1979. Predicate path expressions. In *Proceedings of the 6th. ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL '79)*. ACM Press, New York, NY, 226-236. DOI:https://doi.org/10.1145/567752.567774

[3] Ian Editor (Ed.). 2007. *The title of book one* (1st. ed.). The name of the series one, Vol. 9. University of Chicago Press, Chicago. DOI:https://doi.org/10.1007/3-540-09237-4.

[4] David Kosiur. 2001. *Understanding Policy-Based Networking* (2nd. ed.). Wiley, New York, NY..

[5] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning. Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (2017). DOI:https://doi.org/10.1145/3133956.3134015

[6] Rakesh Bahadur Yadav, P Santosh Kumar, and Sunita Vikrant Dhavale. 2020. A Survey on Log Anomaly Detection using Deep Learning. 2020 8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO) (2020). DOI:https://doi.org/10.1109/icrito48877.2020.9197818

[7] Xiaoyun Li, Pengfei Chen, Linxiao Jing, Zilong He, and Guangba Yu. 2020. SwissLog: Robust and Unified Deep Learning Based Log Anomaly Detection for Diverse Fults. 2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE) (2020). DOI:https://doi.org/10.1109/issre5003.2020.00018

[8] Mengying Wang, Lele Xu, and Lili Guo. 2018. Anomaly Detection of System Logs Based on Natural Language Processing and Deep Learning. 2018 4th International Conference on Frontiers of Signal Processing (ICFSP) (2018). DOI:https://doi.org/10.1109/icfsp.2018.8552075

[9] R. Vinayakumar, K. P. Soman, and Prabaharan Poornachandran. 2017. Long short-term memory based operation log anomaly detection. 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI) (2017). DOI:https://doi.org/10.1109/icacci.2017.8125846

[10] Haixuan Guo, Shuhan Yuan, and Xintao Wu. 2021. LogBERT: Log Anomaly Detection via BERT. 2021 International Joint Conference on Neural Networks (IJCNN) (2021). DOI:https://doi.org/10.1109/ijcnn52387.2021.9534113

[11] Hudan Studiawan, Ferdous Sohel, and Christian Payne. 2021. Anomaly Detection in Operating System Logs with Deep Learning-Based Sentiment Analysis. IEEE Transactions on Dependable and Secure Computing 18, 5 (2021), 2136-2148. DOI:https://doi.org/10.1109/tdsc.2020.3037903

[12] Hashemi, S. and Mäntylä, M. 2021. OneLog: Towards end-to-end training in software log anomaly detection. arXiv preprint arXiv:2104.07324 (2021). DOI: https://doi.org/10.48550/arXiv.2104.07324

[13] Xiaoyu Duan, Shi Ying, Wanli Yuan, Hailong Cheng, and Xiang Yin. 2021. QLLog: A log anomaly detection method based on Q-learning algorithm. Information Processing & Management 58, 3 (2021), 102540. DOI:https://doi.org/10.1016/j.ipm.2021.102540

[14] Yali Yuan, Sripriya Srikant Adhatarao, Mingkai Lin, Yachao Yuan, Zheli Liu, and Xiaoming Fu. 2020. ADA: Adaptive Deep Log Anomaly Detector. IEEE INFOCOM 2020 - IEEE Conference on Computer Communications (2020). DOI:https://doi.org/10.1109/infocom41043.2020.9155487

[15] Thorsten Wittkopp, Alexander Acker, Sasho Nedelkoski, Jasmin Bogatinovski, Dominik Scheinert, Wu Fan, and Odej Kao. 2022. A2Log: Attentive Augmented Log Anomaly Detection. Proceedings of the Annual Hawaii International Conference on System Sciences (2022). DOI:https://doi.org/10.24251/hicss.2022.234

[16] Ermal Elbasani and Jeong-Dong Kim. 2021. LLAD: Life-Log Anomaly Detection Based on Recurrent Neural Network LSTM. Journal of Healthcare Engineering 2021, (2021), 1-7. DOI:https://doi.org/10.1155/2021/8829403

[17] Van-Hoang Le and Hongyu Zhang. 2021. Log-based Anomaly Detection Without Log Parsing. 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE) (2021). DOI:https://doi.org/10.1109/ase51524.2021.9678773

[18] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I. Jordan. 2009. Detecting large-scale system problems by mining console logs. Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles - SOSP (2009). DOI:https://doi.org/10.1145/1629575.1629587

[19] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R. Lyu. 2017. Drain: An Online Log Parsing Approach with Fixed Depth Tree. 2017 IEEE International Conference on Web Services (ICWS) (2017). DOI:https://doi.org/10.1109/icws.2017.13

[20] Adetokunbo A.O. Makanju, A. Nur Zincir-Heywood, and Evangelos E. Milios. 2009. Clustering event logs using iterative partitioning. Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD (2009). DOI:https://doi.org/10.1145/1557019.1557154

[21] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. Proceedings of the IEEE 86, 11 (1998), 2278-2324. DOI:https://doi.org/10.1109/5.726791

[22] Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, and Michael R. Lyu. 2019. Tools and Benchmarks for Automated Log Parsing. 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP) (2019). DOI:https://doi.org/10.1109/icse-seip.2019.00021

[23] Pinjia He, Jieming Zhu, Shilin He, Jian Li, and Michael R. Lyu. 2016. An Evaluation Study on Log Parsing and Its Use in Log Mining. 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN) (2016). DOI:https://doi.org/10.1109/dsn.2016.66

[24] Shilin He, Jieming Zhu, Pinjia He, and Michael R. Lyu. 2020. Loghub: A Large Collection of System Log Datasets towards Automated Log Analytics. Arxiv (2020). DOI:https://doi.org/ https://doi.org/10.48550/arXiv.2008.06448

[25] Siyang Lu, Xiang Wei, Yandong Li, and Liqiang Wang. 2018. Detecting Anomaly in Big Data System Logs Using Convolutional Neural Network. 2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech) (2018).

DOI:https://doi.org/10.1109/dasc/picom/datacom/cyberscitec.2018.00037

[26] Weihang Jiang, Chongfeng Hu, Shankar Pasupathy, Arkady Kanevsky, Zhenmin Li, and Yuanyuan Zhou. 2009. Understanding Customer Problem Troubleshooting from Storage System Logs. USENIX Association (2009).

[27] Crimson Consulting Group. The solaris 10 advantage: Under- standing the real cost of ownership of red hat enterprise linux. In Crimson Consulting Group Business White Paper, 2007

[28] The Association of Support Professionals. Technical Support Cost Ratios. 2000

[29] Shilin He, Jieming Zhu, Pinjia He, and Michael R. Lyu. 2016. Experience Report: System Log Analysis for Anomaly Detection. 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE) (2016). DOI:https://doi.org/10.1109/issre.2016.21

[30] Logparser. Retrieved April 22, 2022 from https://logparser.readthedocs.io/

[31] tf–idf - Wikipedia. Retrieved April 22, 2022 from https://en.wikipedia.org/wiki/Tf%E2%80%93idf