Log Files Based Prediction of Failure Types using Deep Learning Models

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering

in

Computer Science and Engineering

Submitted By

Nidhi Patel

(Roll No. 801632031)

Under the supervision of

Dr. R.K. Sharma

Professor, CSED

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY
PATIALA – 147004

July 2018

CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled, ***"Log Files Based Prediction of Failure Types using Deep Learning Models"***, in partial fulfillment of the requirements for the award of degree of Master of Engineering in *Computer Science and Engineering* submitted in Computer Science and Engineering Department of Thapar Institute of Engineering and Technology, Patiala, is an authentic record of my own work carried out at SanDisk India under the supervision of Senior Manger *Mr. Abhinav Anand* and guide Dr. *R.K. Sharma* and refers other researcher's work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

Nidhi Patel

(Nidhi Patel)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

R K Sharma

(R.K. Sharma) 13.7.18

Professor, CSED

ACKNOWLEDGEMENT

The successful completion of any task would be incomplete without acknowledging the people who made it possible and whose consistent direction and encouragement secured success As a matter of first importance, I might want to offer my thanks to **Mr. Abhinav Anand, Senior Manager, Software Tools Team, SanDisk India Device Design center Pvt. Ltd.** for all his guidance and support. This work was empowered and maintained by his vision and thoughts. Secondly, I would like to express my gratitude to my guide **Dr. R.K. Sharma, Professor, Thapar Institute of Engineering & Technology, Patiala** who gave me the flexibility to investigate without anyone else and in the meantime the direction to recoup when my means vacillated. His patience and support helped me overcome many crisis situations and finish this thesis. He has been wellspring of motivation to me.

I am grateful to **Dr. Maninder Singh, Head of Department** and **Dr. Ashutosh Mishra, ME-CSE Coordinator, Department of Computer Science & Engineering** and entire faculty and the friends who devoted their valuable time and helped me in all possible ways towards successful completion of this thesis work. I thank all those who have contributed directly or indirectly to this work. I would like to express my heartfelt thanks to my friends who with their thought provoking vies, veracity and whole heated co-operation helped me in doing this dissertation.

Lastly, I would like to thank my family for their years of unyielding love and encouragement. They have always wanted the best for me and I appreciate their determination and sacrifice.

Nidhi Patel

(Nidhi Patel)

ABSTRACT

In order to verify the correct working of a firmware, a firmware engineer writes various test cases, out of which some tests can get failed due to various issues. Firmware engineer finds the reasons behind the failure of test cases based on logs generated against the test cases, so that they can be resolved and proper verification of firmware can be done. Thus this log files based prediction of a failure types system is developed which is capable of identifying the patterns in the log files that are generated for a test across regression by a firmware engineer and able to classify it in a particular failure type. Error logs are an important source of information both for diagnosis as well as for proactive test failure handling. This log file based prediction includes different stages, namely, Pre-processing, Feature Extraction and Prediction. In Pre-processing phase, regular expressions are written to remove all the non- alphanumeric characters (noise) from logs and to extract test statistics and error description as this is the valuable piece of information or pattern that will help in prediction of failure type. Feature extraction is a process in which we try to extract only that relevant information from the pre-processed data. It can be a low level feature or high level feature. Different classifiers have been used to predict the failure type. This thesis presents a Deep Learning model based prediction system which is trained using Adaptive Moment Estimation (Adam), a Stochastic Optimization algorithm for parameters updation. The Deep Learning models, namely, Convolutional Neural Network (CNN), Recurrent Neural Network-LSTM (RNN-LSTM), and Recurrent Convolutional Neural Network (RNN-CNN) are analogous to neural networks having learnable weights and biases which evaluates log vector to final winner class scores. A comprehensive recognition rate of 87.0% using CNN, 84.0% using RNN-LSTM and 84.0% using RNN-CNN is achieved on a set of 10,000 logs.

Contents

List of Tables

List of Figures

Abbreviations

1. Deep Learning - DL
2. Convolutional Neural Network - CNN
3. Recurrent Convolutional Neural Network - LSTM - RNN - LSTM
4. Recurrent Convolutional Neural Network – RNN-CNN
5. Support Vector Machines - SVMs
6. Adaptive Moment Estimation – Adam
7. Convolution layer 1 - C1
8. Convolution layer 2 - C2
9. Subsampling layer 1 - S1
10. Subsampling layer 3 - S3
11. RNN Layer 1 - R1
12. Fully Connected layer 4 - FC4
13. Actual Failure Type Prediction – AFT
14. Predicted Failure Type - PFT

CHAPTER 1
INTRODUCTION

1.1 Log Files Based Prediction of Failure Types

Log files based prediction of failure types is a system that is capable of identifying the patterns in the log files which are generated for a test across regression by a firmware engineer and able to classify it in a particular failure type.

The ultimate success of an embedded system project depends both on its software and hardware. Firmware engineers develop the software that manages electronic devices and the various steps taken by embedded firmware engineer are listed in Figure 1.

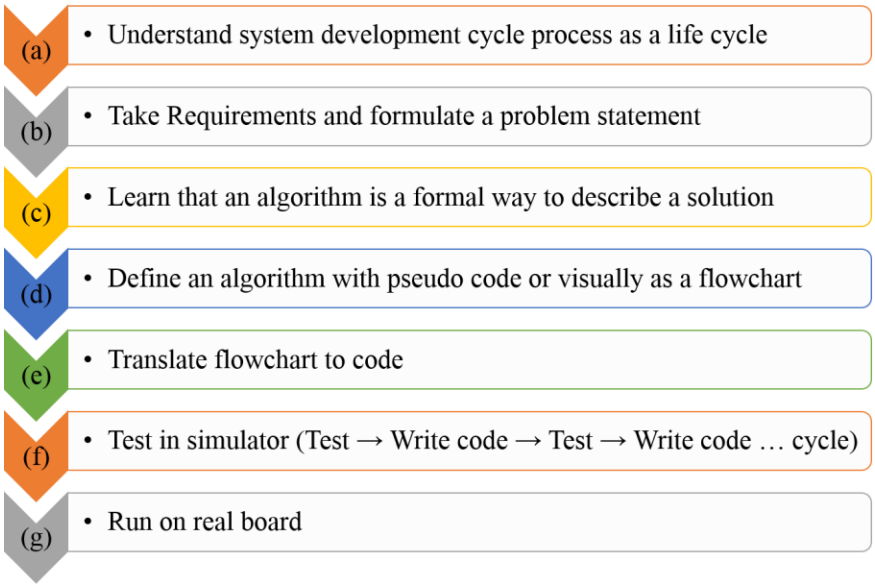

Figure 1. Steps taken by Embedded Firmware Engineer

This system helps a firmware engineer in (f) and (g) steps in order to decode properly the reason behind the failure of a particular test and its type so that it can be assigned to corresponding department to make the suitable changes.

Log files are significant for diagnosis [8] [9] as well as for proactive test failure handling. However elaborate data preparation is necessary to filter out valuable pieces of information. Thus this log file based prediction includes different stages, namely, Pre-processing, Feature

Extraction and Prediction. In Pre-processing phase, regular expressions are written to remove all the non- alphanumeric characters (noise) from logs and to extract test statistics and error description as this is the valuable piece of information or pattern that will help in prediction of failure type. One of the basic pre-processing technique is to identify borders between messages. As error messages travel through various design and architectural levels of the system, several information is aggregated until the resulting log-record is written to the log file which often leads to situations where one log record covers several lines in the file as the original error message is quoted several times within one log record. This duplicate information is eliminated by assigning each piece to a fixed section in the log file such that each line corresponds to exactly one log record. This also involves the usage of a unique field delimiter. Next to the pre-processing phase, a prediction system does the feature extraction. Feature extraction is a process in which we try to extract only the relevant information from the pre-processed data that a classifier needs to classify the dataset into different classes. It can be a low level feature or a high level feature. Once the relevant features are extracted, different classifiers can be used to predict the failure type.

Log files based prediction of a failure type system is broadly classified into two categories – Before Recreating Option Prediction of Failure Type and After Recreating Option Prediction of Failure Type Prediction.

In Before Recreating Option Prediction of Failure Type, the firmware engineer collects the log files of the failed test cases and pass it for further processing and hence the prediction without making any changes or updating the log files. These systems have the phases as shown in Figure 2.

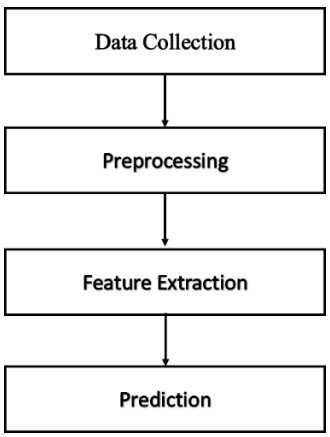

Figure 2. Phases in Before Recreating Option Prediction of Failure Type

In After Recreating Option Prediction of Failure Type, firmware engineer recreates the complete environment of testing similar to regression. This helps in creating the correct logs even in the situations where the test has failed due to system break downs, say, power failure. These modified logs are then used to decide the reasons for failure, *i.e.* failure type before recreating the architecture.

1.2 Log Files Dataset

The 10,000 log files for failed test cases have been collected from SanDisk India Server. Out of these 10,000 files, 8,500 log files correspond to firmware issue, hardware issue, lab issue, model issue and tool issue failure types. Each of these failure types contributed equally in the dataset, considered in this work. Other 1500 logs files were involving test issue failure type. Log files downloaded have the naming convention as follows:

"Name of the test" + "timestamp" + "failure type".

The log files follow a definite structure. The description of the structure is given below:

- The log starts with the test header that consists of information regarding the name of the test lab name, project name, specification of simulator or hardware.
- After the test header ends, the test statistics block starts which consists of information like total read and write in GB.
- When the test header ends, command history block starts. This block consists of executed commands of the particular test.
- After the command history ends, the error header block starts that consists of the error description and this block is the end of a log file.

A hypothetical data looks like this:

11:41:02.718 ::Info : # Test Statistics Start #
Connected : 0.157199772246
To provider : 0.15319903564
11:41:02.749 ::Info : # Test Statistics End #
11:41:02.749 ::Info : # Error Header Start #
11:41:10.427 ::Info : Traceback (most recent call last):
"Address IP_OCTET maps to URL, but this does not map back to the address - POSSIBLE BREAK-IN ATTEMPT!"

1.3 Deep Learning Models in Log Files Based Classification

Deep Learning (DL) models exhibit a good performance in analyzing the log data [17] [19]. It consists of good computational power and automatically extracts the features required for the solution of the problem. Three Deep Learning algorithms have been used for developing log files based failure type prediction system owing to the following facts:

1. Convolutional Neural Network: Convolution is a mathematical operation that operates on two functions (*g* and *h*) in order to produce a third function. It is typically viewed as the amount by which the original function is translated by finding the integral of the pointwise multiplication of two functions. One of the method of applying convolution is by moving a window to our given input data and letting the neural network learn the weights to apply to adjacent words and thus a solution for not just embedding unique words, but also bigrams (word pairs), trigrams (word triplets), *etc*. [18].
2. Recurrent Neural Network-LSTM: Traditional methods like Support Vector machines (SVMs), Decision Tree, Random Forest *etc*. fails in determining the semantic relationship between the words. Thus a, Deep Learning algorithm Recurrent Neural Network-LSTM can be used for the Log Analysis [19] as it consists of a hidden layer which acts as a memory and stores the internal state of the log data. Whenever the new data arrives, the memory gets updated, and decisions are made based on the current and previous input.
3. Recurrent Convolutional Neural Network: This deep learning model is a combination of Convolution and Recurrent layers. This can be used as a variant in order to get the better results.

The detailed explanation of the above Deep learning Models is given in the section below.

1.3.1 Convolutional Neural Network (CNN)

The Convolution Neural Network (CNN) comprises of 3 types of layers: Convolution Layer, Max Pooling Layer and Fully Connected Layer [11] [12]. Different topologies of all these layers are designed to build a CNN. Each layer processes the 3D input and transforms it to 3D output through a differentiable function.

Each Convolution layer consists of learnable filters that pass through full height and width of the 3D input. As this learnable filter extends through the 3D input, it computes the dot product between the values of a raw pixel image and a filter values as shown in Figure 3 [14]. It produces a 2D activation map of that filter. All the 2D outputs produced along the height and width of the input, and along with the depth are stacked together to produce full output of the layer.

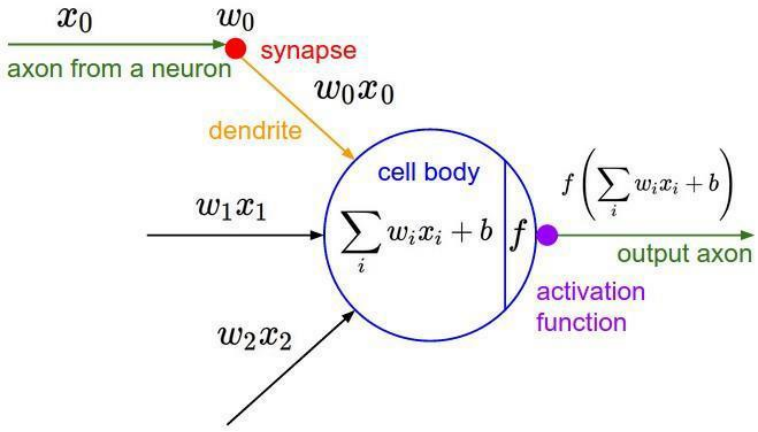

Figure 3. Working of Neuron

Local Connectivity: When working with the 3D input volume, it is impossible to connect all neurons to every previous layer's neuron. Instead each neuron is connected to a small unit. The extent of this unit is called as receptive field.

For example, if we have an input of size - 32×32×3 and receptive field is of size - 5×5, then every neuron will have 5×5×3 connections to the input as shown in Figure 4 [14].

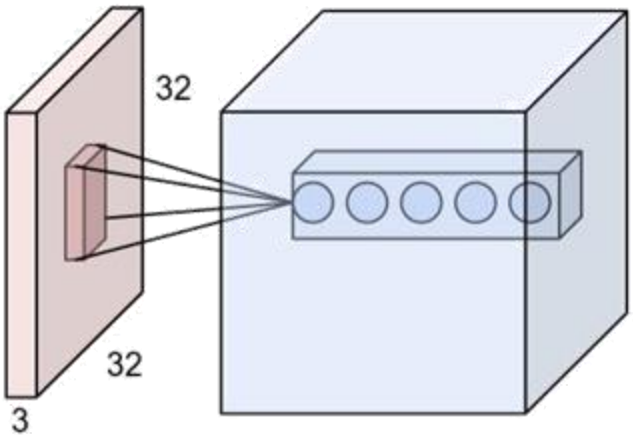

Figure 4. Local Connectivity

Spatial Arrangement: This arrangement explains how many neurons are there in output and how they are arranged. There are 3 things that control the size of output volume. These are: Depth, Stride and Zero-padding.

i. Depth: This controls the number of neurons in the output volume that will connect to same input. For example, if the convolution layer takes image as input, then different neurons will activate along the depth of the input depending on the different color blobs of image. The set of neurons, working on the same unit of input, are referred as Depth Column.

ii. Stride: It is defined as the step that is taken, when the filter is operating on the unit of the input along the height and width. For example, if we take stride as 1, the filter will jump along height/width with one step only. Higher stride value will result in less overlap and the output will be of small size. Conversely, lower stride results in heavy overlap and output generated is of bigger size.

iii. Zero Padding: This feature also controls the output volume size. Suppose w is size of the input, f is size of receptive field and s is stride value then, output volume size (O) is calculated as

$$O = [(w - f + 2 \times p) / s] + 1$$

Where p is size of the zero padding used.

For example, Output size (O) for input of size (w) = 5, stride (s) = 1 with zero padding (p) =1 is [(5 - 3 + 2) / 1] +1 = 5. Figure 5 [14] explains the output.

Neuron with receptive field (f) of 3

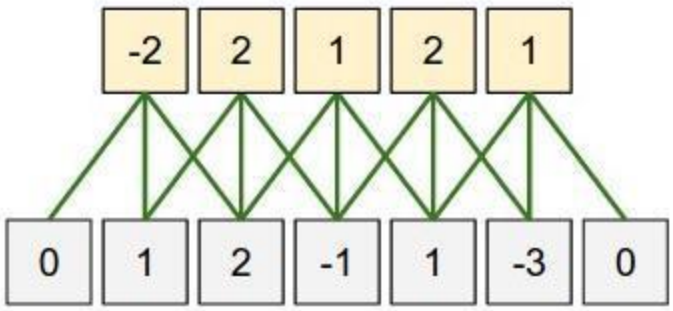

Figure 5. Dot Product of Input Vector and Receptive Field Vector with Stride =1

For the same input but stride (s) = 2, output size (O) is [(5 - 3 + 2) / 2] +1 = 3. Figure 6 [14] explains the output.

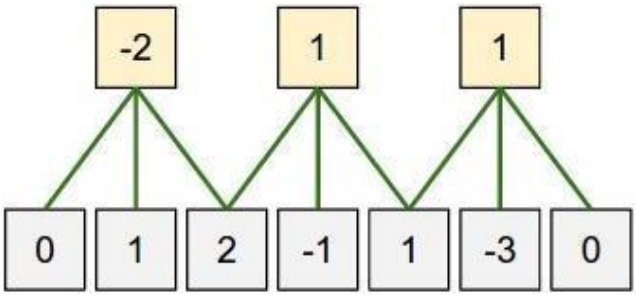

Figure 6. Dot Product of Input Vector and Receptive Field Vector with Stride = 2

Sub-Sampling Layer: Task of this layer is to reduce the size of input which helps in controlling over fitting. It reduces the resolution of the image as shown in Figure 7 [14].

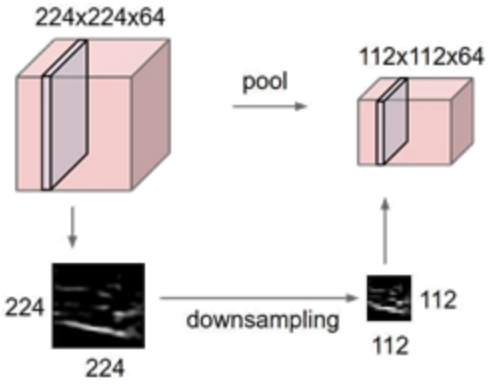

Figure 7. Working of Sub-Sampling layer

A Pooling layer with filter size 2×2, down samples the input by 2 along the height and width of the input as shown in Figure 8 [14].

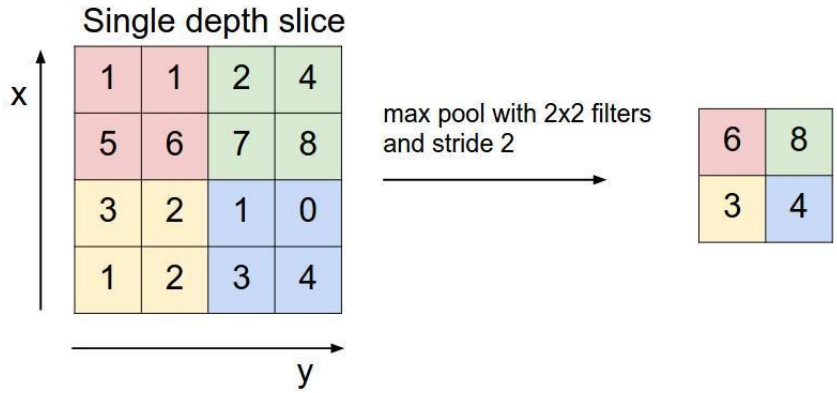

Figure 8. Downsampling of 4x4 Vector

Fully Connected layer: This layer's neurons are connected to all the neurons of the previous layer. The output is computed as the matrix multiplication followed by addition of learnable bias to it.

Adaptive Moment Estimation (Adam) Algorithm

Gradient based learning algorithm are of 2 types – Batch gradient and Stochastic gradient. In Batch gradient algorithms, parameters are updated once the exact gradients are calculated from the entire training dataset after training process. In stochastic mode, parameters are updated on the basis of a noisy gradient that is calculated using few samples of training dataset. These few samples are randomly selected from the entire dataset. Stochastic mode is faster as compared to batch mode. Adam – a Stochastic Optimization Algorithm (variant of Stochastic Gradient Descent) is believed to converge faster [1] [13] as compared to the traditional version and is used to update network weights iterative based in training data without incurring additional cost. It has little memory requirements, appropriate for sparse data and also corrects the ill-functioning of the loss function. Owing to these reasons, we decided to train our system using stochastic learning mode.

1.3.2 Recurrent Neural Network – LSTM (RNN-LSTM)

Recurrent Neural Networks are the state of the art algorithm for sequential data. RNN deals with sequence information. RNN makes use of available Sequence information. RNNs are called recurrent because they perform the same task for every element of a sequence by that it means that we have a "memory" which captures information about what has been calculated so far and pass that memory to the next node. Availability of memory helps in understanding the context of the sequence. RNN models are used widely used in NLP tasks because of their ability to handle sequences.

Normal Neuron: Normal neuron takes in some input, it can be multiple input so it can aggregate them and then once it aggregate those inputs it passes through some sort of activation function(RELU function in above example) and then an output is generated as shown in Figure 9 [15].

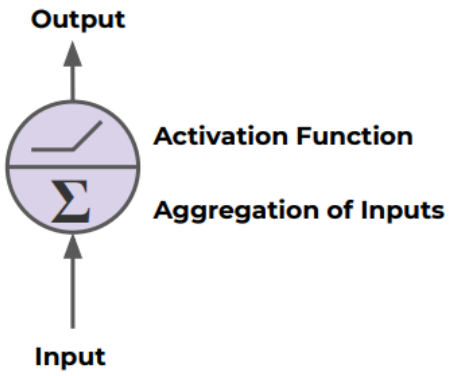

Figure 9. Normal Neuron

Recurrent Neuron: In Recurrent neural network the output goes back into input of the same neuron. So we can actually unroll it throughout time as shown in Figure 10 [15].

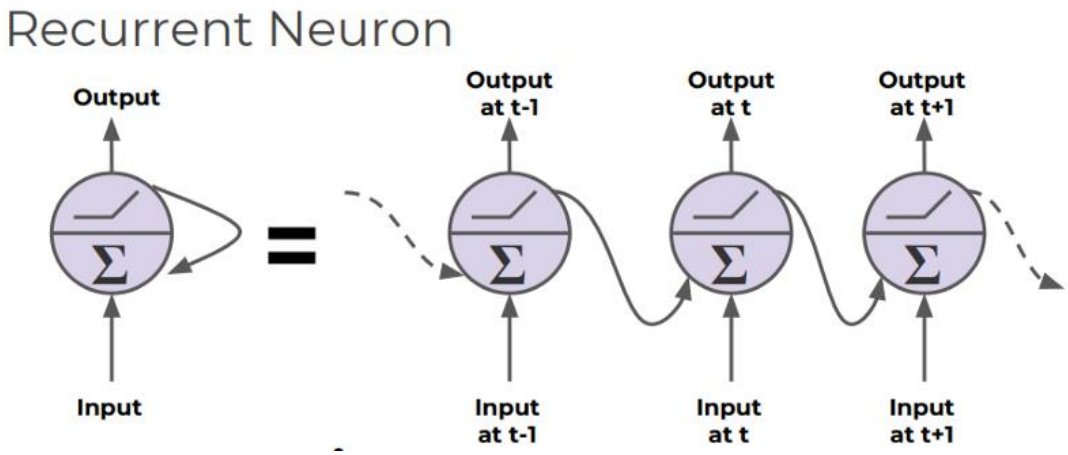

Figure 10. Recurrent Neuron

Time is represented as t-1, t and t+1 and we have this particular neuron where it's input at t-1 gives output t-1 and that gets pass into a neuron in its state input at time t and that has an output at time t and then we can take that output and pass it as input for the same neuron at time t+1 and then so on.

Memory Cells: Cells that are a function of inputs from previous time steps are also known as memory cells.

Recurrent Neural Networks: The input X goes through recurrent neurons, output y is generated and then this output y is passed it back in all the neurons as shown in Figure 11 [15].

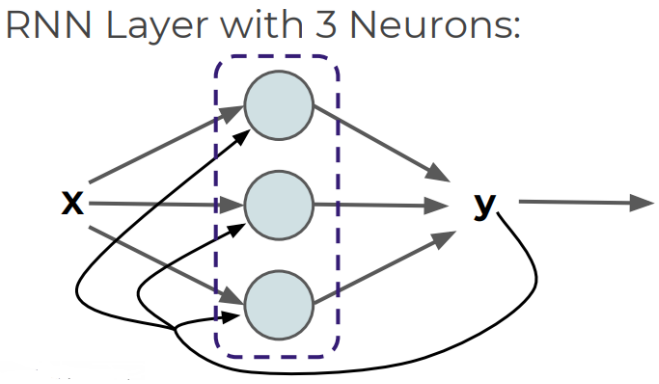

Figure 11. Recurrent Neural Network Layer

When unrolling this entire layer throughout time we get the network as shown in the Figure 12 [15].

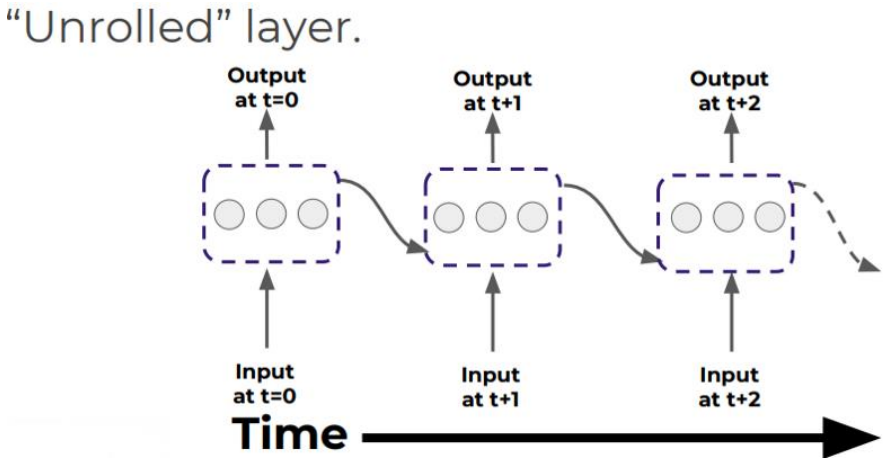

Figure 12. Recurrent Neural Network Layer when unrolled thought time

When we unroll the entire layer through time we get an input at t=0 and an output at t=0 and pass that along with output and input at t+1. It has same idea as the single neuron just difference we are doing it for the entire layer. Now since the output of these recurrent neuron at a time t is technically a function of all the inputs from previous time step so we can begin to think that it has some form of memory because we are technically passing in historical information into that layer of recurrent neurons.

Example 1: We can perform a sequence input and sequence output and example of that will be a time series information such as years' worth of daily sales data and then wanting the same sales data shifted over a certain time period in future as shown in Figure 13 [15].

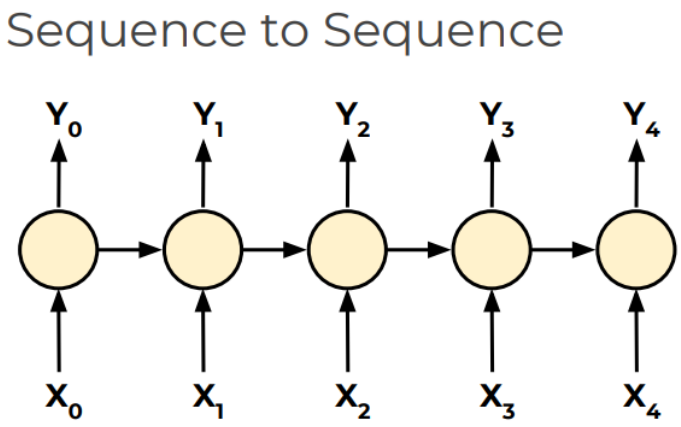

Figure 13. Example of sequence input and sequence output

Example 2: This type of recurrent neural network providing a sequence input and request back a vector output will be used in sentiment analysis. In which you feed a sequence of word or a paragraph of a movie review and request back output indicating whether it was a positive sentiment or negative as shown in Figure 14[15].

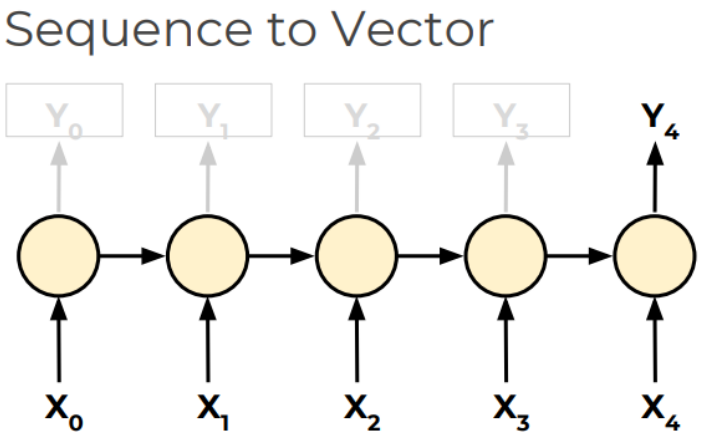

Figure 14. Example of sequence input and vector output

Example 3: In this recurrent neural network you feed in a vector input at first time steps and basically pass zeroes for rest of time steps and then let output be a sequence so that's a vector to sequence network. This type of Recurrent Neural Network can be used in Image Captioning where you feed in a single image and request back a sequence words describing that image as shown in Figure 15 [15].

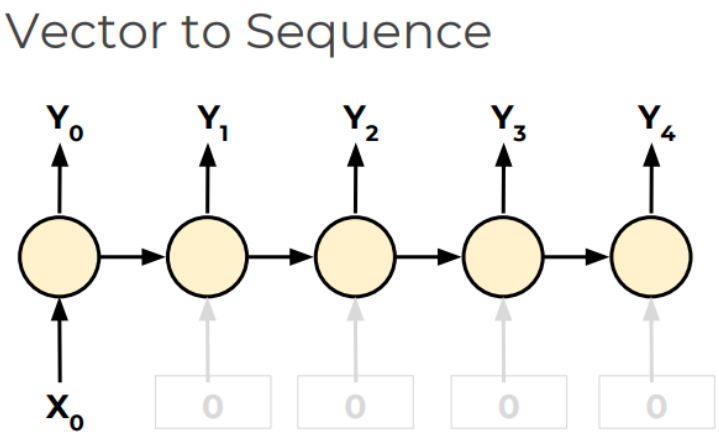

Figure 15. Example of vector input and vector output

1.3.3 Recurrent Convolutional Neural Network (RNN-CNN)

This deep learning model is a combination of Convolution and Recurrent layers [10]. Thus RNN-CNN can be used a variant in order to get the better results.

1.4 Summary of Thesis

This thesis work is organized into six chapters. Each chapter is written to be largely self-contained and complete. The summary of each chapter is given below.

In chapter 1, the log files based prediction of failure types using Deep Learning models is introduced. This system is capable of predicting the failure type against the failed test cases written by firmware engineer to verify the correct working of the firmware either on simulator or on real board based on log files created for failed test cases across regression using Deep Learning models, namely, Convolutional Neural Network, Recurrent Neural Network-LSTM, and Recurrent Convolutional Neural Network. This system has two phases *i.e.* before recreating option prediction and after recreating option prediction. Then, the log files dataset is discussed followed by reasons for using the above three Deep Learning models. After this, a detailed explanation for each Deep Leaning models is presented.

In chapter 2, a review of the literature on preprocessing, feature extraction and classification techniques is carried out in the field of log files based applications. This chapter provides a way to find the research gap and to propose an idea for thesis work which is discussed in the chapter 3.

In chapter 3, the motivation and the research objectives of this thesis are laid down. On the basis of the objectives, the problem and complete methodology is formulated in various subparts which is discussed in the chapter 4.

In chapter 4, various steps are discussed to build the log files based failure type prediction system. A complete methodology including data preparation, feature extraction and an algorithm for training different classifiers using Adam optimization algorithm method is discussed.

In chapter 5, an experimental setup is described. This chapter includes the obtained experimental outcomes and related discussions.

In chapter 6, a series of conclusions, concerning the experiments performed and the results obtained are drawn and the possible future investigations are also indicated.

This thesis work ends with a set of references and Appendices that aim precisely on log files analysis based application using Machine Learning and Deep Learning models.

CHAPTER 2
LITERATURE REVIEW

Log files based prediction, detection or classification is an important issue in pattern recognition and a decent measure of automation is required for this task. Bags of pre-processing, feature extraction and classification techniques have been developed in past decades and calculable work has been done for log files analysis based applications. Following is the state of the art in the different phases of the prediction.

2.1 Review on Pre-processing of Log Files

We have numerous steps to follow before we pass the data to classifier. Various algorithms and techniques have been proposed for pre-processing of the data which help to reduce the noise, identify the pattern and increase the prediction results. According to Salfner *et al.* (2012), most commonly used pre-processing steps are eliminating log file rotation, identifying borders between messages using unique field delimiters, converting timestamp of log in which machine can process. While writing records to a log file, this log file rotation technique switches a log to a new log file if the current log file has reached a size limit, time span limit, or both. Data has to be reorganized in order to form one large chronologically ordered log file. As the error messages "travel" through various modules and structural levels of the system, more and more information is collected until the resulting log-record is written to the log file. This often leads to situations where one log record covers several lines in the file as the original error message is quoted several times within one log record. Also, Bertero *et al.* (2017) explains the technique to remove all the non-alphanumeric characters from the logs in order to smoothly identify the patterns in logs and for further feature engineering. Du *et al.* (2017) parsed the unstructured, free-text log entries into a structured representation to learn a sequential model over this structured data.

2.2 Review on Feature Extraction from Log Files

After pre-processing next step is feature engineering. Various feature extraction techniques have evolved in past few years. Each log file has its own features, so feature extraction techniques are also different for all the log files. There is no standard that a particular feature extraction technique will give good results for a given log file. Fulp *et al.* (2015) forms a vector from Sys log file describing Message M as (1:1, 6:2, 30:1, 37:1, 38:2, 78:2, 189:1), where each value is the tag: count and used this vector to classify M as belonging to a fail or non-fail system. Gurudimma *et al.* (2015) formed the feature matrix by extracting the counters and running jobs on each node from the resource usage logs using regular expressions. Juneja *et al.* (2016) forms the feature vector by extracting the Trace ID, Actor, Timestamp and Activity from event log files using the regular expressions. Du *et al.* (2017) extracted a "logkey" (also known as "message type") from each log entry of system log file based on the idea of LCS (longest common subsequence). Bertero *et al.* (2017) proposed various methods to obtain relevant features from the log files like Count Vectors as features, TF-IDF Vectors as features, Word level, N-Gram level, Character level, Word Embedding's as features, NLP based features.

2.3 Review of Classification Technique on Log Files

Salfner *et al.* (2012) proposed three algorithms namely, assignment of error IDs to error messages based on Levenshtein's edit distance, clustering approach to group similar error sequences, and statistical noise filtering algorithm. They performed various experiments on data of a commercial telecommunication system to build the model for an accurate error-based online failure prediction and measured accuracy of predictions in terms of F-measure. Maximum F-measure of 0.66 is achieved.

Gurudimma *et al.* (2015) used Principle Component Analysis (PCA) to increase the error handling time window using Resource Usage Logs by 10.0%.

Fulp *et al.* (2015) described a Support Vector Machines (SVMs) based approach to predict hard disk failure using system log files (Syslogs). They achieved an accuracy of 73.0%.

Juneja *et al.* (2016) worked on improving the Goodness (fitness and structural complexity) of the process models by clustering structurally similar traces by splitting the event-log into homogeneous subsets. They implemented the K-Medoid clustering algorithm with two different distance metrics namely, Longest Common Subsequence (LCS) and Dynamic Time Warping (DTW) and were successful in proposing the algorithm that returns the cluster set with highest goodness ratio of 2.49×10^{-03} to automate the clustering process.

Bertero *et al.* (2017) proposed the word embedding technique, a Google's word2vec algorithm based approach for minimizing the human intervention for log files processing technique and exploited a feature space using standard classifiers. They performed several experiments and validated their approach by seeking stress patterns using Naïve Bayes', Random Forest and Neural Networks classifiers. They achieved the highest accuracy of 90.0%.

Du *et al.* (2017) proposed an automatic log pattern learning system based on deep neural network model using Long Short-Term Memory (LSTM) and detect anomalies when log patterns deviate from the model trained from log data under normal execution. They have reported accuracy of 97.0% in their work.

CHAPTER 3

PROBLEM STATEMENT

The firmware engineer writes various tests to validate or verify the firmware. These tests can fail due to various issues like failure of model or hardware on which the tests are running or due to the faulty code written by firmware engineer itself.

Thus, whenever any test case gets failed a log containing various information like error description, simulator information, test statistics *etc*. is generated. Now based on this log information firmware engineer finds the reason behind the failure of a particular test known as failure type, so that it can be assigned to different department to make required changes.

As of now, this failure type prediction is being done by a firmware engineer based on recognition of suitable patterns from log files and assign the issue to the corresponding department.

A good amount of work has been done, based on log files using classical machine learning algorithms like SVMs, Naïve Bayes, Neural networks *etc*. and have been applied in various industrial applications. However, deep learning models like convolutional Neural Network and Recurrent Convolutional Network have not been used on log files based applications.

Thus, the present study is undertaken with a focus on developing an approach using Deep Learning Models to predict the failure types based on SanDisk India industrial Logs generated for a test across regression. First step is to parse the log and secondly to find some pattern (like Test Statistics, Error Description) which helps to classify it in one of the following six failure types:

- Firmware Issue
- Hardware Issue
- Lab Issue
- Model Issue
- Tool Issue
- Test Issue

CHAPTER 4

LOG FILES BASED FAILURE TYPE PREDICTION SYSTEM

Our log files based prediction of failure types system is trained to predict the failure types based on identifying the patterns in the labelled 10,000 log files which are generated for a test across regression. To build the system we performed the four major tasks as shown in Figure 16. These tasks are Data Preparation, Feature Engineering, Preprocessed data loading and Training the different classifiers.

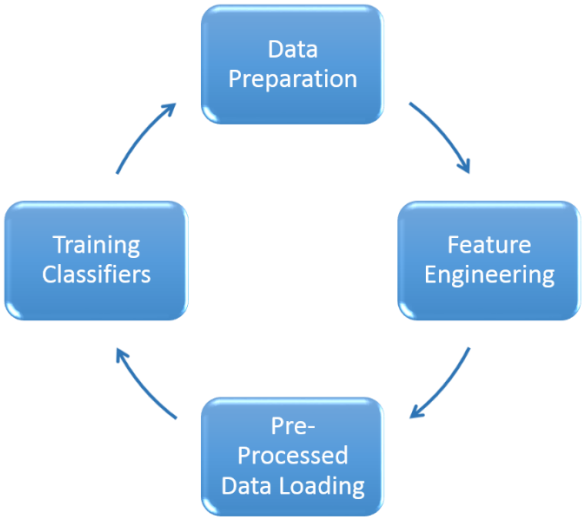

Figure 16. Tasks performed in building the Prediction system

4.1 Data Preparation

Logs are collected from the SanDisk India server and failure types (target) are extracted from the names of the log files. After this regular expressions are written to remove all the non-alphanumeric characters and also regular expressions for extracting "Test Statistics" and "Error Description" patterns from the logs. Using the Error Description a Corpus "C" (text file) is created containing the first column as failure type and second column (separated by space) as Error Description.

Also Test Statistics contain twenty five parameters using which a .csv file is created containing all the twenty five parameters along with failure types label.

Here correspond to each failure type a label is assigned. The different labels assigned to different failure types are:-

F1-Firmware Issue

F2-Hardware Issue

F3-Lab Issue

F4-Model Issue

F5-Tool Issue

F6-Test Issue

Now these labels are used for further processing.

4.2 Feature Engineering

The next stage is the feature engineering. In this step, new features are created using the existing dataset by transforming the raw text data of corpus into feature vectors. Here, count vectors as features algorithm is implemented in order to obtain relevant features from our dataset.

4.2.1 Count Vectors as Features

Count Vector is a vector notation of the dataset in which every row represents a document (error description) from the corpus, every column represents a term from the corpus, and every cell represents the frequency count of a particular term in a particular document. The method of creating count vectors is discussed below.

Consider a Corpus S of T documents $\{t_1, t_2 \ldots . t_T\}$ from which R unique tokens are extracted [16] [20]. Using these R tokens, a dictionary is formed and the size of the Count Vector matrix V is computed using $T \times R$ where T represents the number of rows and R represents the number of columns. Each entry in the matrix V represents the frequency of tokens in document T (i). Example given below will clear the concept more accurately.

T1: Ram is a good boy. Seeta is also good.

T2: Laxman is a good person.

T3: Kirti is an intelligent person.

The dictionary created is a list of unique tokens (words) in the corpus T.

['Ram', 'Seeta', 'Kirti', 'good', 'intelligent', 'boy', 'Laxman', 'person']

Here, T=3, R=8

The count matrix V of size 3 × 8 will be represented as –

	Ram	Seeta	Kirti	good	intelligent	boy	Laxman	person
T1	1	1	0	2	0	1	0	0
T2	0	0	0	1	0	0	1	1
T3	0	0	1	0	1	0	0	1

Now, in this count matrix V of size 3 × 8 each column R corresponds to the tokens in the dictionary and represents the word vector for the corresponding word in the matrix V. For example, the word vector for 'Ram' in the above matrix is [1, 0, 0] and so on. Similarly, each row T represents the number of documents. Here, count matrix consist of 3 rows since the number of documents are 3. The third row in the above matrix V is read as – T3 contains 'Kirti': once, 'intelligent': once and 'person': once.

Each error description is considered as one document in the Corpus. Thus, by applying Count Vector feature algorithm we are able to extract thousands of unique words. Out of which, we selected only fifty words based on frequency and then prepared a dictionary. In contrast, if all thousand words have been considered then the output matrix would contain more number of zeros *i.e.* it would be very sparse and therefore inefficient for any computation.

Now we have numerical representation of error description and in a format where Deep Learning architectures are capable of processing it. A final data.csv file with 10,000 instances with 75 features (50 from error description and 25 from test statistics) corresponds to 10,000 logs is created where each row corresponds to each log file known as Log vector and first fifty columns contains the numerical representation of error description and rest twenty five contains parameters from the test statistics.

4.3 Pre-Processed Data Loading

The dataset data.csv file is loaded and is spliced into training and testing sets so that classifier can be trained and tested.

Since, the input to a network is extremely wide (1x75), an embedding layer is added in order to reduce the dimensionality of the input.

4.3.1 Batch size

Total dataset is divided in batch size of 100, thus a total of 100 batches are created.

4.4 Training the Network

After data preparation, converting it into a format in which deep learning models can process and loading the preprocessed data with batch size, the final step is to train the deep learning classifiers and test them against the test dataset.

4.4.1 Training Convolutional Neural Network (CNN)

CNN has 3 types of layers: Convolution layer, Subsampling layer, Fully Connected layer arranged in following topology.

Input layer > Convolution layer (C1) > Subsampling layer (S2) > Fully Connected layer (FC4) > output layer.
The complete architecture of the network is shown in Figure 17.

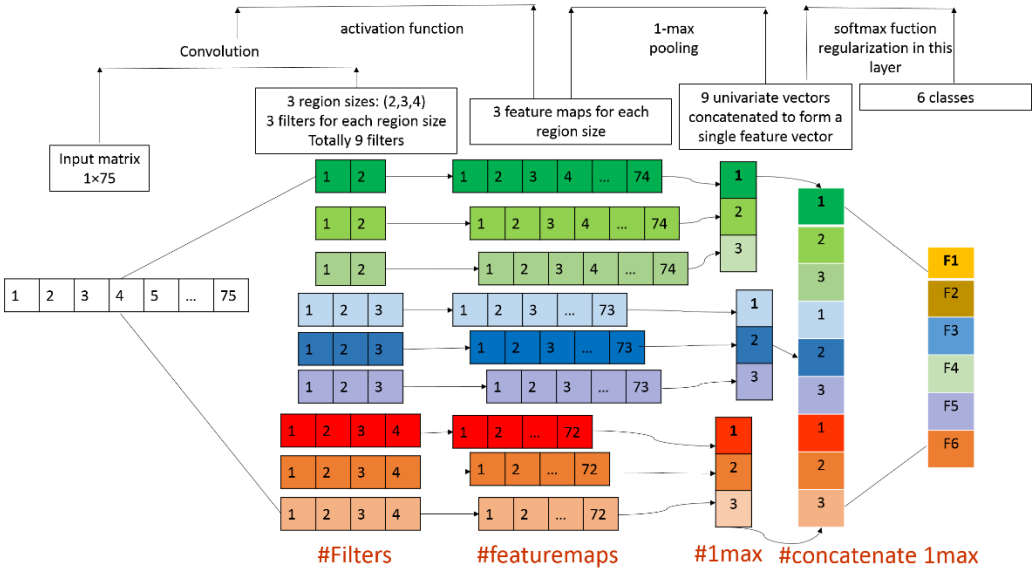

Figure 17. CNN Architecture

Each layer contains trainable weights and biases. The input layer takes input from 1×75 vector.

The first convolution layer - C1 has 9 feature maps, each of three of sizes 1×74, 1×73, 1×72 respectively with stride 1. Each 1×2, 1×3, 1×4 unit of the input layer is connected to 1×2, 1×3, 1×4 respective unit of C1 layer. This 1×2, 1×3 and 1×4 receptive field is multiplied by trainable coefficient and at last passed through the rectified linear function.

The second sub sampling layer - S2 with 9 feature maps of univariate size down samples the output of C1 layer. A unit of 1×74, 1×73, and 1×72 of each of three feature maps in C1 layer is connected to each feature map of S2 layer. The nine univariate feature maps are then concatenated to form a single feature map.

The fourth fully connected layer – FC4 has 6 units and is fully connected to S2 layer. The input vector is multiplied to weight vector and bias is added. The output is then passed through a sigmoid squashing function.

The whole network is trained using Adam Stochastic Optimizatiom technique as it converges faster as compared to any gradient based technique without incurring any additional cost [1] [13] on our dataset of 10,000 log vectors. The different experiments are performed based on

the split of the dataset. We keep on increasing the epochs until we get the convergence in test accuracy.

4.4.2 Training Recurrent Neural Network (RNN) – LSTM

RNN has 3 types of layers arranged in following topology.
Input layer > RNN Layer (R1) > output layer.

Each layer contains trainable weights and biases. The input layer takes input from 1×75 vector. Thus input layer will have 75 neurons.

The first RNN layer - R1 has 5 neurons. This RNN layer will corresponds to 10,000 outputs from t=0 to t=9,999. The input vector is multiplied to weight vector and bias is added. The output is then passed through a sigmoid function.
The output of t=0 will be fed to an input at t+1, t+1 output to input at t+2 and so on. Also only the last six time stamp outputs i.e. from t_{9994} to t_{10000} are considered for the final output.

The whole network is trained using Adam Stochastic Optimizatiom technique as it converges faster as compared to any gradient based technique without incurring any additional cost [1] [13] on our dataset of 10,000 log vectors. The different experiments are performed based on the split of the dataset. We keep on increasing the epochs until we get the convergence in test accuracy.

4.4.3 Training Recurrent Convolutional Neural Network (RNN-CNN)

Recurrent Convolutional Neural Network has 3 types of layers arranged in following topology. Input layer > RNN Layer (R1) > Convolution layer (C2) > Subsampling layer (S3) > Fully Connected layer (FC4)> output layer.

Each layer contains trainable weights and biases. The input layer takes input from 1×75 vector. Thus input layer will have 75 neurons.

The first RNN layer - R1 has 5 neurons. This RNN layer will corresponds to 10,000 outputs from t=0 to t=9,999. The input vector is multiplied to weight vector and bias is added. The output is then passed through a sigmoid function.

The output from last 75 neurons will be an input to a Convolution layer C2 that has 9 feature maps, each of three of sizes 1×74, 1×73, 1×72 respectively with stride 1. Each 1×2, 1×3, 1×4 unit of the input layer is connected to 1×2, 1×3, 1×4 respective unit of C1 layer. This 1×2, 1×3 and 1×4 receptive field is multiplied by trainable coefficient and at last passed through the rectified linear function.

The second sub sampling layer - S2 with 9 feature maps of univariate size down samples the output of C1 layer. A unit of 1×74, 1×73, and 1×72 of each of three feature maps in C1 layer is connected to each feature map of S2 layer. The nine univariate feature maps are then concatenated to form a single feature map.

The fourth fully connected layer – FC4 has 6 units and is fully connected to S2 layer. The input vector is multiplied to weight vector and bias is added. The output is then passed through a sigmoid squashing function.

The whole network is trained using Adam Stochastic Optimizatiom technique as it converges faster as compared to any gradient based technique without incurring any additional cost [1] [13] on our dataset of 10,000 log vectors. The different experiments are performed based on the split of the dataset. We keep on increasing the epochs until we get the convergence in test accuracy.

CHAPTER 5

EXPERIMENTS AND RESULTS

We performed several experiments in this work. A total of twelve experiments are performed using three Deep Learning algorithm CNN (Strategy A), RNN-LSTM (Strategy B), and RNN-CNN (Strategy C) on the dataset consisting of 75 features and 10,000 instances. Table 1 shows the statistics on these experiments.

Table 1. Experimental Setup

Experiment #	Strategy Used	# of training instances	# of testing instances
1	A	9,000	1,000
2	B	9,000	1,000
3	C	9,000	1,000
4	A	8,000	2,000
5	B	8,000	2,000
6	C	8,000	2,000
7	A	7,000	3,000
8	B	7,000	3,000
9	C	7,000	3,000
10	A	6,000	4,000
11	B	6,000	4,000
12	C	6,000	4,000

5.1 Training the System using 9,000 instances

Experiment – 1

In first experiment, we trained the system for 9,000 log vectors and tested for 1,000 log vectors using strategy A *i.e.* CNN. Training of the system is done till a convergence in accuracy is obtained. In this experiment, system attained the convergence after 7 epochs and shows constant test accuracy for next 3 epochs. For the first 7 epochs, test accuracy increases from 74.4% to 79.0%. Table 2 shows accuracies on train and test dataset. Figure 18 shows the training and testing accuracy for each epoch.

Table 2. Training and Testing Accuracies for Experiment 1

Epoch	Training Accuracy (%)	Test Accuracy (%)
1	89.9	74.4
2	91.5	75.7
3	92.9	76.4
4	94.6	77.0
5	98.9	77.9
6	99.2	78.4
7	100.0	79.0
8	100.0	79.0
9	100.0	79.0
10	100.0	79.0

The test accuracy of 79.0% means 790 out of 1000 log vectors are correctly classified. Table 3 shows the confusion matrix of the test dataset.

Experiment - 2

In second experiment, we trained the system for 9,000 log vectors and tested for 1,000 log vectors using RNN-LSTM. Training of the system is done till a convergence in accuracy is obtained. In this experiment, system attained the convergence after 6 epochs and shows constant test accuracy for next 4 epochs. For the first 6 epochs, test accuracy increases from 73.2% to 77.0%. Table 2 shows accuracies on train and test dataset. Figure 18 shows the training and testing accuracy for each epoch.

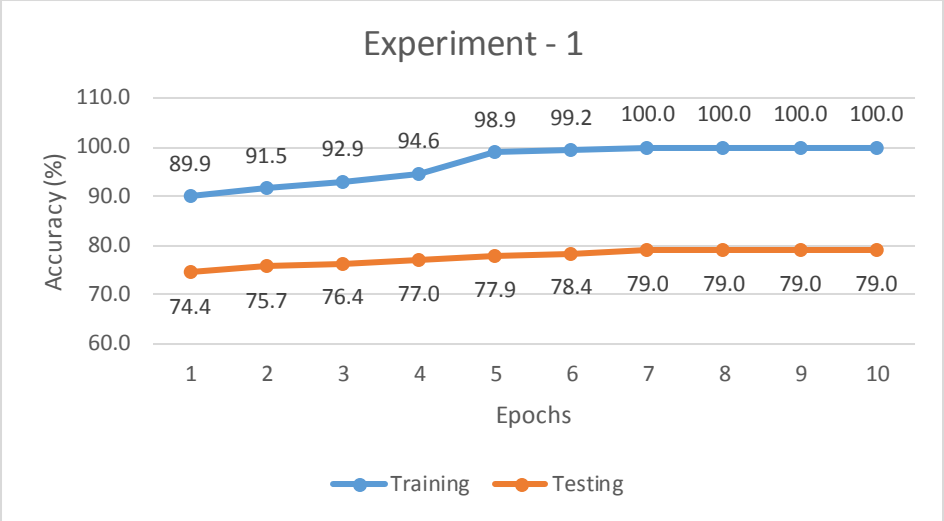

Figure 18. Classification rate on Train and Test dataset of Experiment 1

Table 3. Confusion Matrix for Experiment 1

		Predicted Failure Type (PFT)					
		F1	**F2**	**F3**	**F4**	**F5**	**F6**
Actual Failure Type (AFT)	**F1**	140	0	0	0	0	30
	F2	0	150	0	10	10	0
	F3	0	0	130	40	0	0
	F4	0	30	30	110	0	0
	F5	0	35	0	0	135	0
	F6	25	0	0	0	0	125

Table 4. Training and Testing Accuracies for Experiment 2

Epoch	Training Accuracy (%)	Test Accuracy (%)
1	87.0	73.2
2	89.0	74.4
3	92.4	74.9
4	95.0	75.4
5	99.8	76.8
6	100.0	77.0

7	100.0	77.0
8	100.0	77.0
9	100.0	77.0
10	100.0	77.0

The test accuracy of 77.0% means 770 out of 1,000 log vectors are correctly classified. Table 5 shows accuracies on train and test dataset. Figure 19 shows the training and testing accuracy for each epoch.

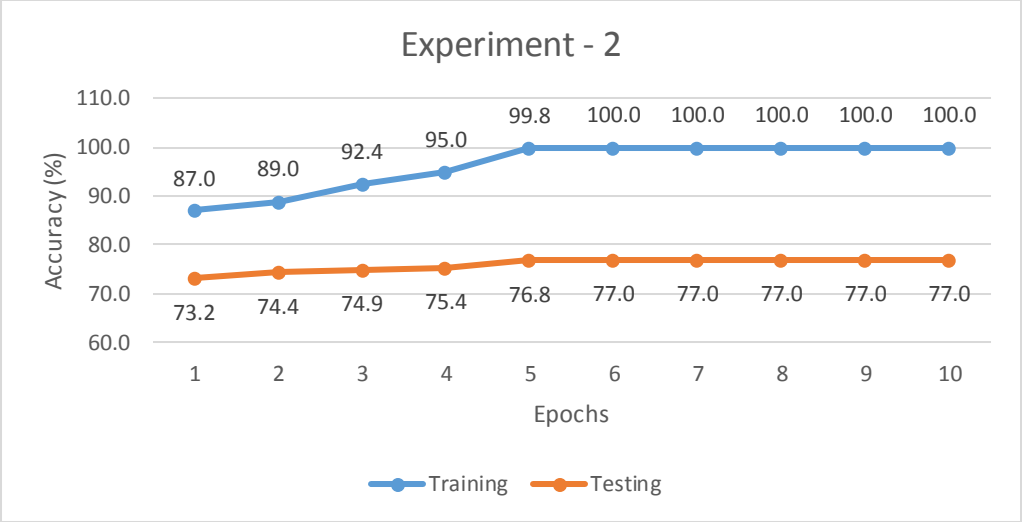

Figure 19. Classification rate on Train and Test dataset of Experiment 2

Table 5. Confusion Matrix for Experiment 2

		Predicted Failure Type (PFT)					
		F1	**F2**	**F3**	**F4**	**F5**	**F6**
Actual Failure Type (AFT)	**F1**	150	0	0	0	0	20
	F2	0	110	0	30	25	5
	F3	0	0	120	50	0	0
	F4	0	10	25	140	0	0
	F5	0	40	0	0	130	0
	F6	30	0	0	0	0	120

Experiment - 3

In third experiment, we trained the system for 9,000 log vectors and tested for 1,000 log vectors using RNN-CNN. Training of the system is done till a convergence in accuracy is obtained. In this experiment, system attained the convergence after 6 epochs and shows constant test accuracy for next 4 epochs. For the first 6 epochs, test accuracy increases from 74.4% to 77.0%. Table 6 shows accuracies on train and test dataset. Figure 20 shows the training and testing accuracy for each epoch.

Table 6. Training and Testing Accuracies for Experiment 3

Epoch	Training Accuracy (%)	Test Accuracy (%)
1	86.0	74.4
2	88.8	75.2
3	92.0	75.8
4	95.9	76.2
5	97.8	76.5
6	100.0	77.0
7	100.0	77.0
8	100.0	77.0
9	100.0	77.0
10	100.0	77.0

The test accuracy of 77.0% means 770 out of 1,000 log vectors are correctly classified. Here the accuracy achieved is same as compared to experiment 2 but number of correctly instances of firmware issue has increased and hardware issue is not misclassified into test issue. Table 7 shows the confusion matrix of the test dataset

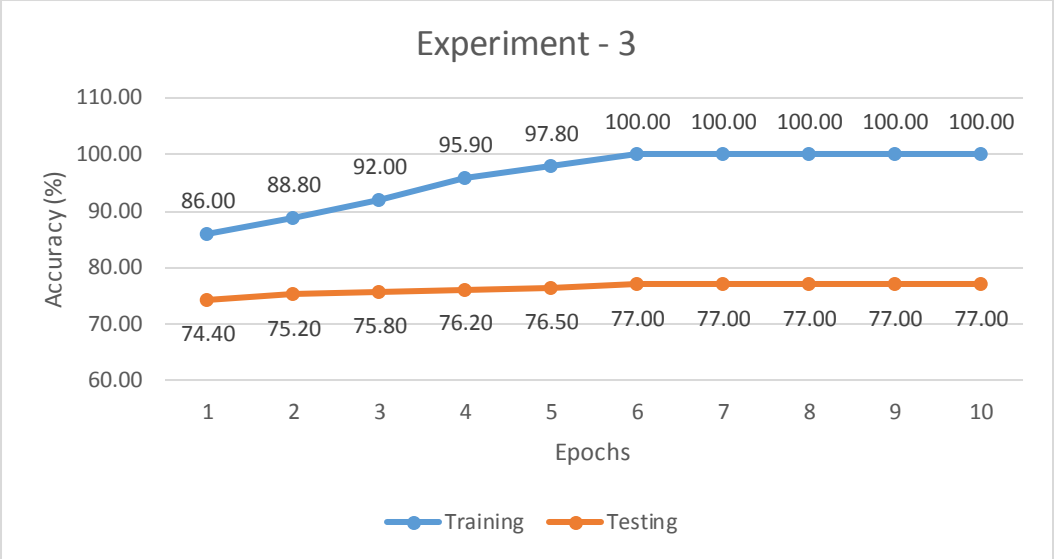

Figure 20. Classification rate on Train and Test dataset of Experiment 3

Table 7. Confusion Matrix for Experiment 3

		Predicted Failure Type (PFT)					
		F1	**F2**	**F3**	**F4**	**F5**	**F6**
Actual Failure Type (AFT)	**F1**	160	0	0	0	0	10
	F2	0	135	0	10	25	0
	F3	0	0	130	40	0	0
	F4	0	20	45	105	0	0
	F5	0	30	0	0	140	0
	F6	50	0	0	0	0	100

After performing the experiment 1, 2 and 3 by splitting the dataset into 90-10 ratio we achieved the accuracy of 79.0%, 77.0% and 77.0% using CNN, RNN-LSTM and RNN-CNN algorithm respectively. The comparative analysis of results for experiment 1, 2 and 3 is shown in Figure 21.

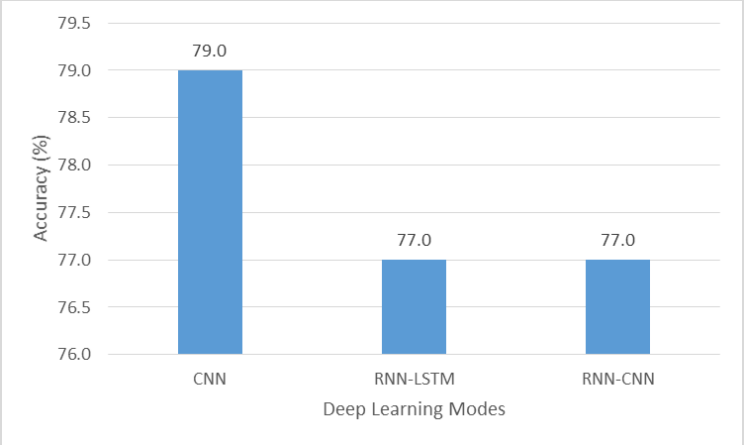

Figure 21. A comparative Analysis of Experiment 1, 2 and 3

The further experiments are performed with an aim to find the optimal split of the dataset. Thus 80-20, 70-30 and 60-40 partition of the dataset is done and experiments are performed. The results of these experiments are listed below.

5.2 Training the System using 8,000 instances

Experiment - 4

In fourth experiment, we trained the system for 8,000 log vectors and tested for 2,000 log vectors using CNN. Training of the system is done till a convergence in accuracy is obtained. In this experiment, system attained the convergence after 7 epochs and shows constant test accuracy for next 3 epochs. For the first 7 epochs, test accuracy increases from 77.4% to 84.0%. Table 8 shows accuracies on train and test dataset. Figure 22 shows the training and testing accuracy for each epoch.

Table 8. Training and Testing Accuracies for Experiment 4

Epoch	Training Accuracy (%)	Test Accuracy (%)
1	89.9	77.4
2	91.2	78.0
3	92.0	78.8
4	92.4	80.4
5	95.6	82.0
6	97.8	83.1

7	98.2	83.8
8	100.0	84.0
9	100.0	84.0
10	100.0	84.0

The test accuracy of 84.0% means 1680 out of 2000 log vectors are correctly classified. Table 9 shows the confusion matrix of the test dataset.

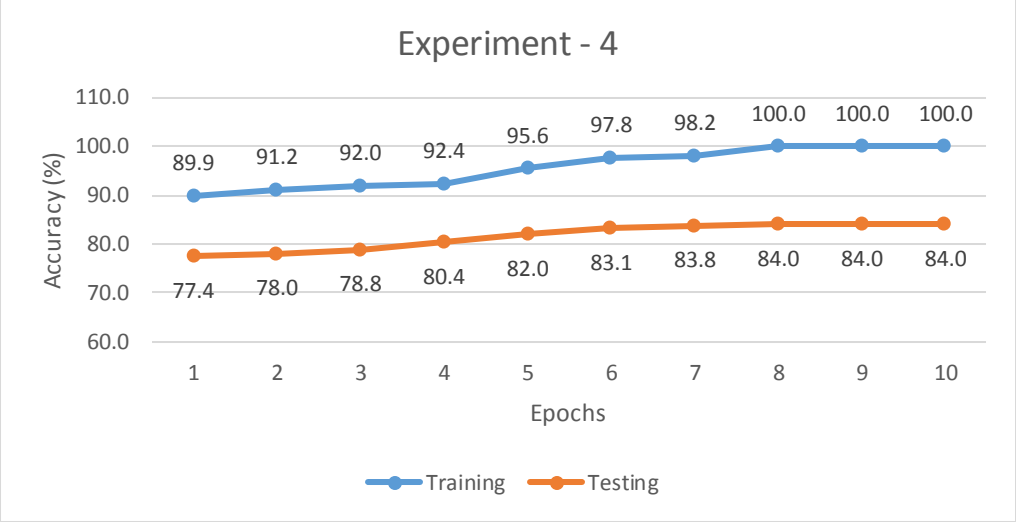

Figure 22. Classification rate on Train and Test dataset of Experiment 4

Table 9. Confusion Matrix for Experiment 4

		Predicted Failure Type (PFT)					
		F1	**F2**	**F3**	**F4**	**F5**	**F6**
Actual Failure Type (AFT)	**F1**	300	0	0	0	0	40
	F2	0	280	0	20	40	0
	F3	0	0	310	30	0	0
	F4	0	0	70	270	0	0
	F5	0	90	0	0	250	0
	F6	30	0	0	0	0	270

Experiment - 5

In fifth experiment, we trained the system for 8,000 log vectors and tested for 2,000 log vectors using RNN-LSTM. Training of the system is done till a convergence in accuracy is obtained. In this experiment, system attained the convergence after 7 epochs and shows constant test accuracy for next 3 epochs. For the first 7 epochs, test accuracy increases from 64.8% to 82.4%. Table 10 shows accuracies on train and test dataset. Figure 23 shows the training and testing accuracy for each epoch.

Table 10. Training and Testing Accuracies for Experiment 5

Epoch	Training Accuracy (%)	Test Accuracy (%)
1	82.1	64.8
2	83.9	77.0
3	84.8	77.6
4	88.9	79.0
5	92.6	80.6
6	94.8	81.2
7	96.7	82.4
8	99.4	82.4
9	100.0	82.4
10	100.0	82.4

The test accuracy of 82.4% means 1648 out of 2,000 log vectors are correctly classified. Table 11 shows the confusion matrix of the test dataset.

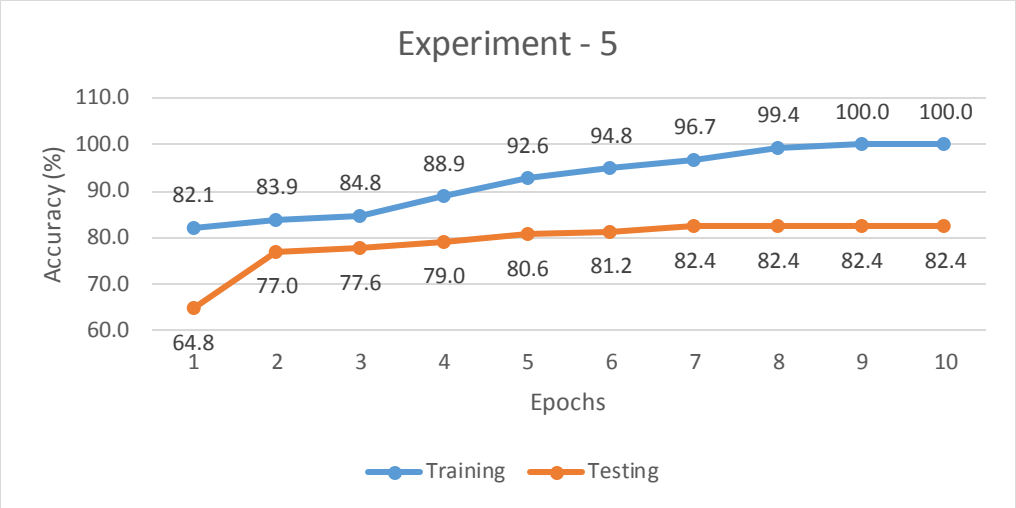

Figure 23. Classification rate on Train and Test dataset of Experiment 5

Table 11. Confusion Matrix for Experiment 5

		Predicted Failure Type (PFT)					
		F1	**F2**	**F3**	**F4**	**F5**	**F6**
Actual Failure Type (AFT)	**F1**	310	0	0	0	0	30
	F2	0	275	0	20	40	5
	F3	0	0	290	50	0	0
	F4	0	27	80	233	0	0
	F5	0	70	0	0	270	0
	F6	30	0	0	0	0	270

Experiment - 6

In sixth experiment for, we trained the system for 8,000 log vectors and tested for 2,000 log vectors using RNN-CNN. Training of the system is done till a convergence in accuracy is obtained. In this experiment, system attained the convergence after 7 epochs and shows constant test accuracy for next 3 epochs. For the first 7 epochs, test accuracy increases from 77.0% to 82.4%. Table 12 shows accuracies on train and test dataset. Figure 25 shows the training and testing accuracy for each epoch.

Table 12. Training and Testing Accuracies for Experiment 6

Epoch	Training Accuracy (%)	Test Accuracy (%)
1	87.4	77.0
2	88.6	77.4
3	89.8	78.8
4	91.7	80.0
5	93.4	80.9
6	95.5	81.4
7	97.4	82.4
8	100.0	82.4
9	100.0	82.4
10	100.0	82.4

The test accuracy of 82.4% means 1648 out of 2000 log vectors are correctly classified. Here the accuracy achieved is same as compared to experiment 5 but number of correctly instances of firmware issue has increased and hardware issue is not misclassified into test issue. Table 13 shows accuracies on train and test dataset. Figure 24 shows the training and testing accuracy for each epoch.

After performing the experiment 4, 5 and 6 by splitting the dataset into 80-20 ratio we achieved the accuracy of 84.0%, 82.4% and 82.4% using CNN, RNN-LSTM and RNN-CNN algorithm respectively. The comparative analysis of results for experiment 4, 5 and 6 is shown in Figure 25.

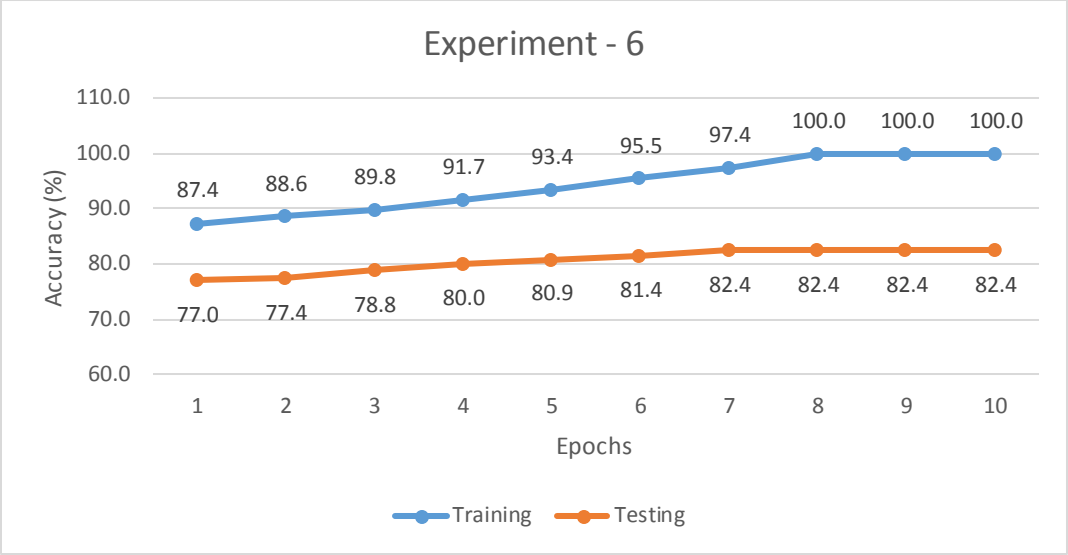

Figure 24. Classification rate on Train and Test dataset of Experiment 6

Table 13. Confusion Matrix for Experiment 6

		Predicted Failure Type (PFT)					
		F1	**F2**	**F3**	**F4**	**F5**	**F6**
Actual Failure Type (AFT)	**F1**	330	0	0	0	0	10
	F2	0	270	0	30	40	0
	F3	0	0	240	100	0	0
	F4	0	32	50	258	0	0
	F5	0	60	0	0	280	0
	F6	30	0	0	0	0	270

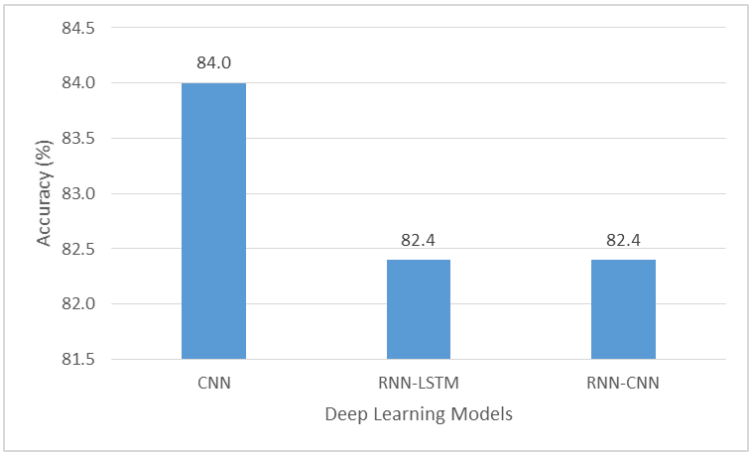

Figure 25. A comparative Analysis of Experiment 4, 5 and 6

Experiments seven, eight, nine splits the dataset in 70-30 ratio. Partition of the dataset is done and experiment is performed. The results are listed below.

5.3 Training the System using 7,000 instances

Experiment - 7

In third experiment, we trained the system for 7,000 log vectors and tested for 3,000 log vectors using CNN. Training of the system is done till a convergence in accuracy is obtained. In this experiment, system attained the convergence after 7 epochs and shows constant test accuracy for next 3 epochs. For the first 7 epochs, test accuracy increases from 80.2% to 87.0%. Table 14 shows accuracies on train and test dataset. Figure 26 shows the training and testing accuracy for each epoch.

Table 14. Training and Testing Accuracies for Experiment 7

Epoch	Training Accuracy (%)	Test Accuracy (%)
1	94.6	80.2
2	95.0	81.7
3	95.3	83.4
4	96.7	84.8
5	96.9	85.9
6	97.2	86.5
7	97.8	87.0
8	98.9	87.0
9	100.0	87.0
10	100.0	87.0

The test accuracy of 87.0% means 2610 out of 3,000 log vectors are correctly classified. Table 15 shows the confusion matrix of the test dataset.

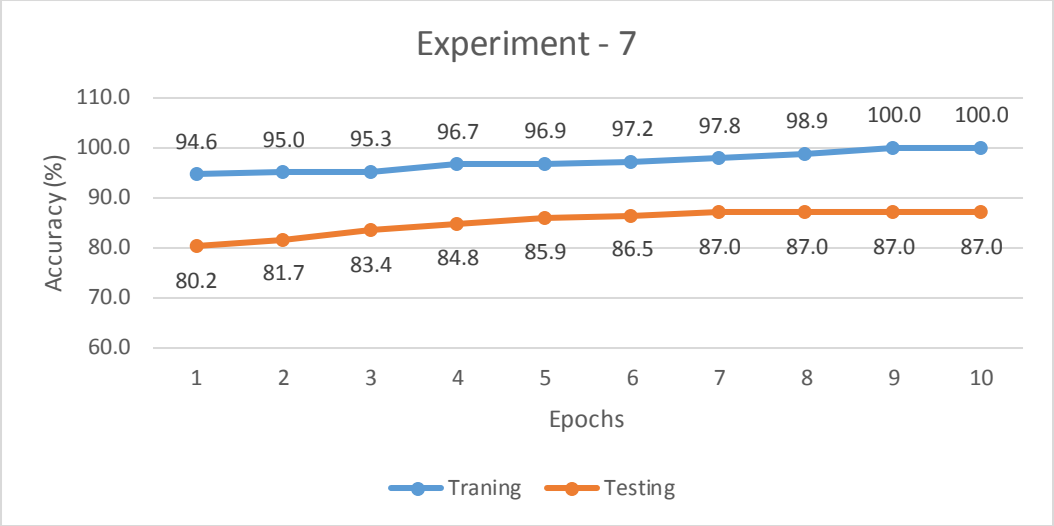

Figure 26. Classification rate on Train and Test dataset of Experiment 7

Table 15. Confusion Matrix for Experiment 7

		Predicted Failure Type (PFT)					
		F1	**F2**	**F3**	**F4**	**F5**	**F6**
Actual Failure Type (AFT)	**F1**	460	0	0	0	0	50
	F2	0	445	0	0	65	0
	F3	0	0	415	95	0	0
	F4	0	20	60	430	0	0
	F5	0	60	0	0	450	0
	F6	40	0	0	0	0	410

Experiment - 8

In third experiment, we trained the system for 70,000 log vectors and tested for using 3000 log vectors using RNN-LSTM. Training of the system is done till a convergence in accuracy is obtained. In this experiment, system attained the convergence after 6 epochs and shows constant test accuracy for next 4 epochs. For the first 6 epochs, test accuracy increases from 79.4% to 84.5%. Table 16 shows accuracies on train and test dataset. Figure 27 shows the training and testing accuracy for each epoch.

Table 16. Training and Testing Accuracies for Experiment 8

Epoch	Training Accuracy (%)	Testing Accuracy (%)
1	88.4	79.4
2	89.2	80.2
3	90.0	80.8
4	92.4	81.9
5	93.8	83.4
6	94.6	84.5
7	95.0	84.5
8	97.9	84.5
9	100.0	84.5
10	100.0	84.5

The test accuracy of 84.5% means 2535 out of 3,000 log vectors are correctly classified. Table 17 shows the confusion matrix of the test dataset.

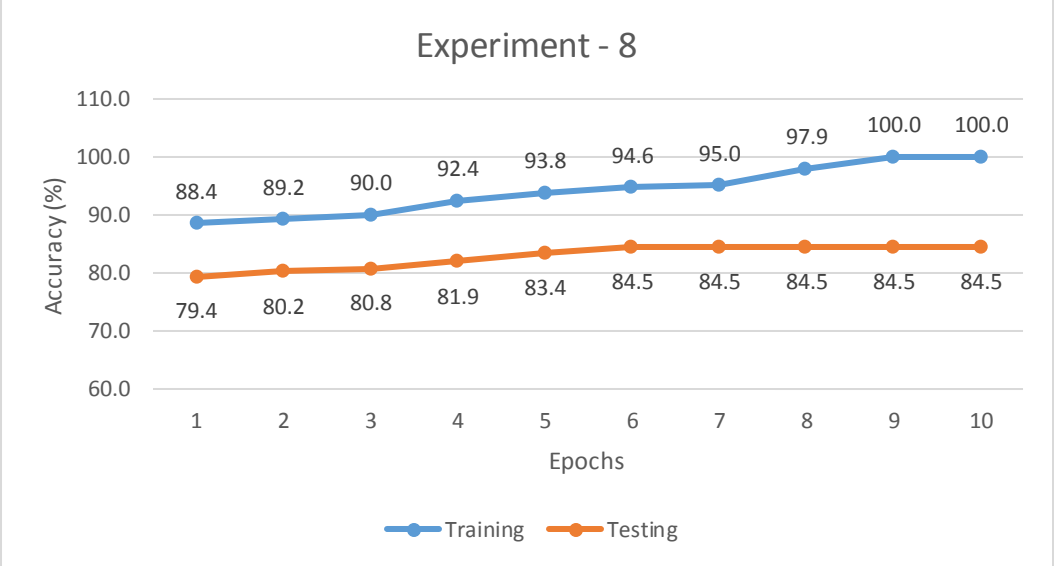

Figure 27. Classification rate on Train and Test dataset of Experiment 8

Table 17. Confusion Matrix for Experiment 8

Actual Failure Type (AFT)	Predicted Failure Type (PFT)						
		F1	**F2**	**F3**	**F4**	**F5**	**F6**
	F1	470	0	0	0	0	40
	F2	0	415	0	30	50	15
	F3	0	0	400	110	0	0
	F4	0	50	70	390	0	0
	F5	0	60	0	0	450	0
	F6	40	0	0	0	0	410

Experiment - 9

In ninth experiment for RNN-CNN, we trained the system for 70,000 log vectors and tested it using 3000 log vectors. Training of the system is done till a convergence in accuracy is obtained. In this experiment, system attained the convergence after 6 epochs and shows constant test accuracy for next 4 epochs. For the first 6 epochs, test accuracy increases from 80.0% to 84.5%. Table 18 shows accuracies on train and test dataset. Figure 28 shows the training and testing accuracy for each epoch.

Table 18. Training and Testing Accuracies for Experiment 9

Epoch	Training Accuracy (%)	Testing Accuracy (%)
1	92.4	80.0
2	93.6	80.4
3	94.2	81.9
4	95.0	82.4
5	95.9	83.8
6	96.3	84.5
7	97.0	84.5
8	97.6	84.5

9	100.0	84.5
10	100.0	84.5

The test accuracy of 72.5% means 2175 out of 3000 log vectors are correctly classified. Here the accuracy achieved is same as compared to RNN-LSTM model but number of correctly instances of firmware issue has increased and hardware issue is not misclassified into test issue. Table 2 shows accuracies on train and test dataset. Figure 16 shows the training and testing accuracy for each epoch.

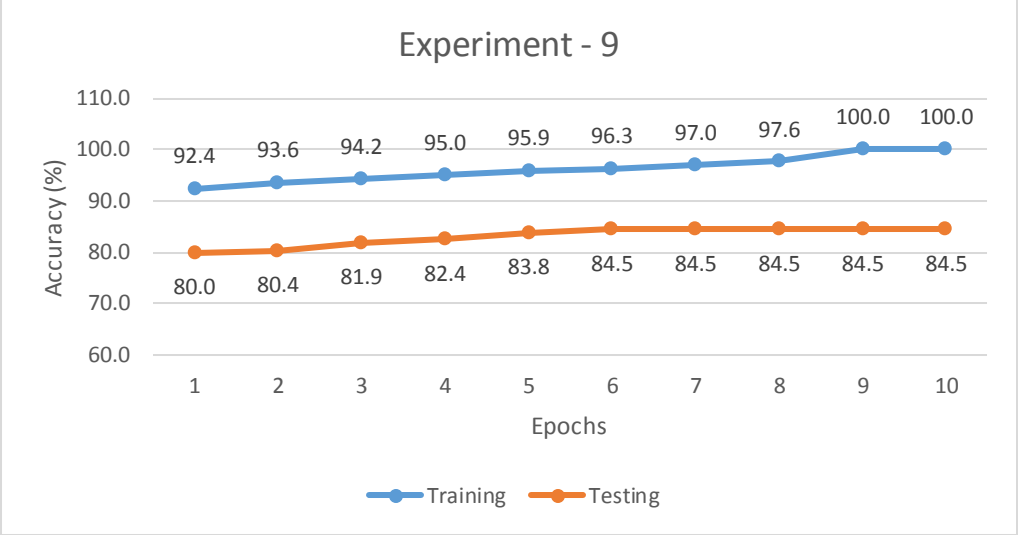

Figure 28. Classification rate on Train and Test dataset of Experiment 9

Table 19. Confusion Matrix for Experiment 9

		Predicted Failure Type (PFT)					
		F1	**F2**	**F3**	**F4**	**F5**	**F6**
Actual Failure Type (AFT)	**F1**	480	0	0	0	0	30
	F2	0	400	0	50	60	0
	F3	0	0	410	100	0	0
	F4	0	35	70	405	0	0
	F5	0	80	0	0	430	0
	F6	40	0	0	0	0	410

After performing the experiment 7, 8 and 9 by splitting the dataset into 70-30 ratio we achieved the accuracy of 87.0%, 84.5% and 84.5% using CNN, RNN-LSTM and RNN-CNN algorithm respectively. The comparative analysis of results for experiment 7, 8 and 9 is shown in Figure 29.

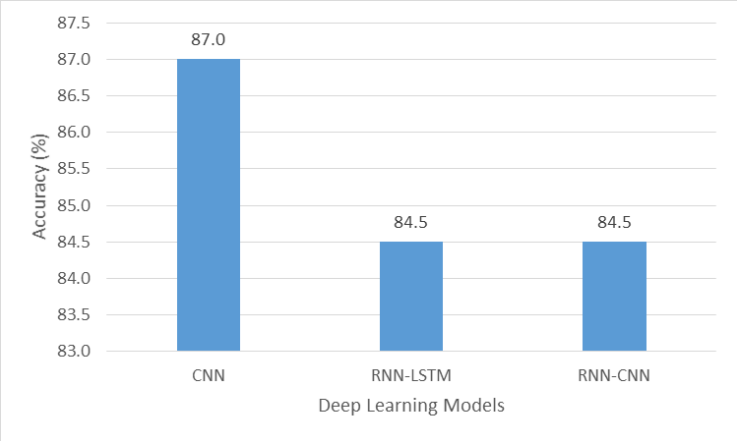

Figure 29. A comparative Analysis of Experiment 7, 8 and 9

Experiments ten, eleven and twelve splits the dataset in 60-40 ratio. Partition of the dataset is done and experiment is performed. The results are listed below.

5.4 Training the System for 6,000 instances

Experiment - 10

In tenth experiment, we trained the system for 6,000 log vectors and tested for using 4,000 log vectors using CNN. Training of the system is done till a convergence in accuracy is obtained. In this experiment, system attained the convergence after 7 epochs and shows constant test accuracy for next 3 epochs. For the first 7 epochs, test accuracy increases from 70.4% to 74.0%. Table 20 shows accuracies on train and test dataset. Figure 30 shows the training and testing accuracy for each epoch.

Table 20. Training and Testing Accuracies for Experiment 10

Epoch	Training Accuracy (%)	Testing Accuracy (%)
1	82.6	70.4
2	84.8	71.2
3	88.9	71.8
4	91.2	72.8

5	92.4	73.2
6	93.6	73.7
7	94.0	74.0
8	94.8	74.0
9	95.2	74.0
10	95.2	74.0

The test accuracy of 74.0% means 2960 out of 4,000 log vectors are correctly classified.

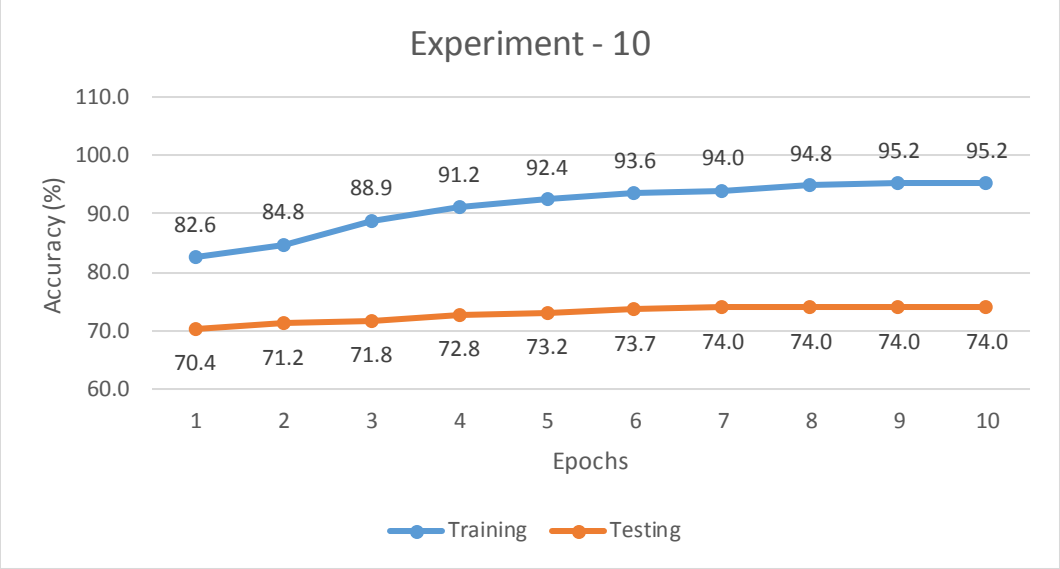

Figure 30. Classification rate on Train and Test dataset of Experiment 10

Experiment - 11

In eleventh experiment, we trained the system for 6,000 log vectors and tested for 4,000 log vectors using RNN-LSTM. Training of the system is done till a convergence in accuracy is obtained. In this experiment, system attained the convergence after 6 epochs and shows constant test accuracy for next 4 epochs. For the first 6 epochs, test accuracy increases from 69.2% to 72.3%. Table 21 shows accuracies on train and test dataset. Figure 31 shows the training and testing accuracy for each epoch.

Table 21. Training and Testing Accuracies for Experiment 11

Epoch	Training Accuracy (%)	Testing Accuracy (%)
1	81.4	69.2
2	82.8	70.4
3	83.6	70.8
4	84.9	71.2
5	85.4	72.1
6	85.8	72.3
7	88.7	72.3
8	91.8	72.3
9	93.7	72.3
10	93.7	72.3

The test accuracy of 72.3% means 2892 out of 4,000 log vectors are correctly classified.

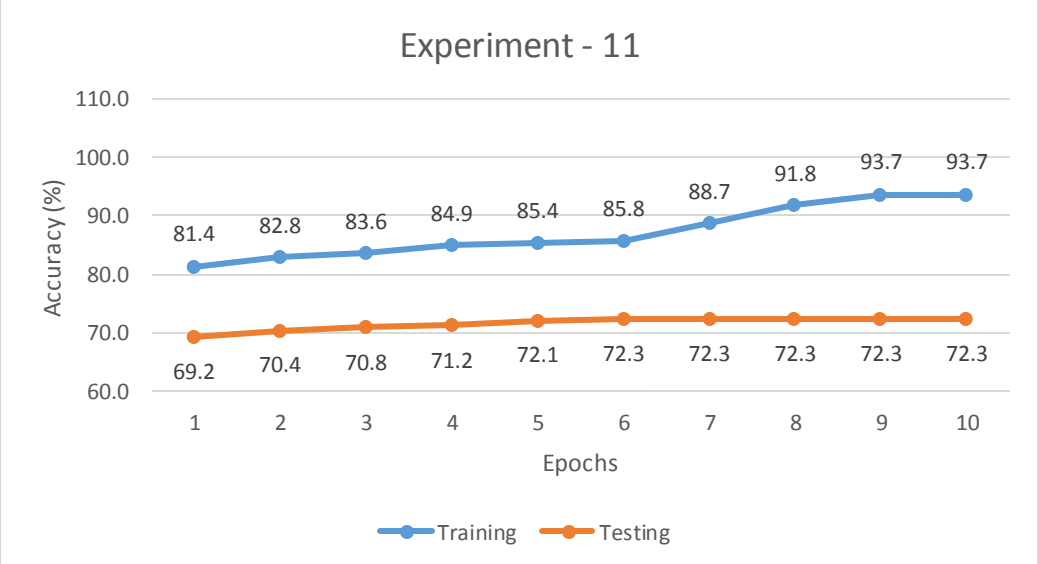

Figure 31. Classification rate on Train and Test dataset of Experiment 11

.

Experiment - 12

In twelfth experiment, we trained the system for 6,000 log vectors and tested for 4,000 log vectors using RNN-CNN. Training of the system is done till a convergence in accuracy is obtained. In this experiment, system attained the convergence after 6 epochs and shows constant test accuracy for next 4 epochs. For the first 6 epochs, test accuracy increases from 69.8% to 72.3%. Table 22 shows accuracies on train and test dataset. Figure 32 shows the training and testing accuracy for each epoch.

Table 22. Training and Testing Accuracies for Experiment 12

Epoch	Training Accuracy (%)	Testing Accuracy (%)
1	81.5	69.8
2	82.4	70.2
3	82.9	70.9
4	83.8	71.4
5	85.7	72.2
6	89.2	72.3
7	92.8	72.3
8	93.7	72.3
9	94.6	72.3
10	94.6	72.3

The test accuracy of 72.3% means 2892 out of 4,000 log vectors are correctly classified.

The accuracy is decreased when dataset is spliced in 60-40 ratio. This is happened because training instances are reduced and thus the model is not trained properly resulting in large number of misclassification of test dataset.

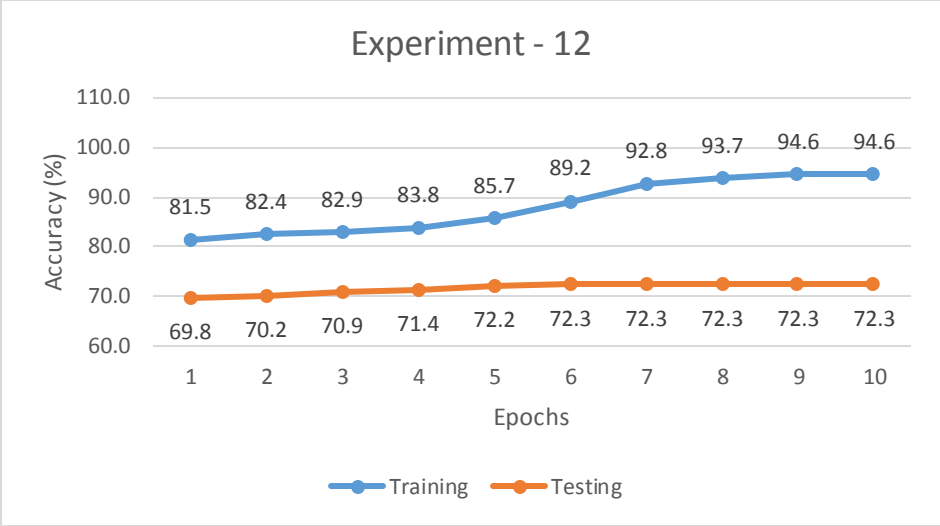

Figure 32. Classification rate on Train and Test dataset of Experiment 12

After performing the experiment 10, 11 and 12 by splitting the dataset into 60-40 ratio we achieved the accuracy of 74.0%, 72.3% and 72.3% using CNN, RNN-LSTM and RNN-CNN algorithm respectively. The comparative analysis of results for experiment 10, 11 and 12 is shown in Figure 33.

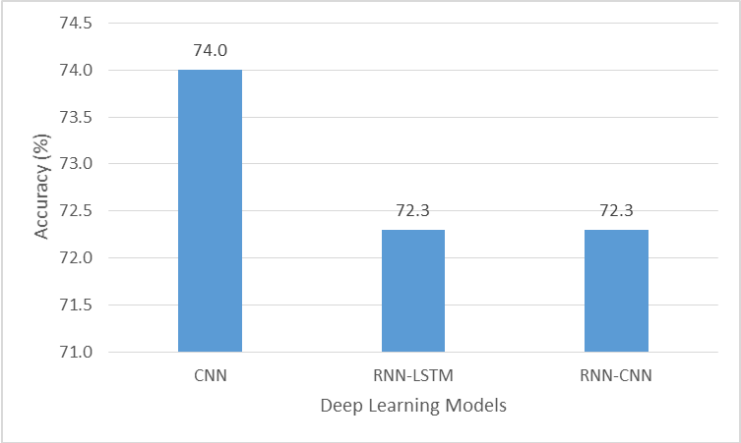

Figure 33. A comparative Analysis of Experiment 10, 11 and 12

From experiments seventh, eighth and ninth this is concluded that 70-30 split is optimal and best accuracy of 87.0% is achieved using CNN.

CHAPTER 6

CONCLUSION AND FUTURE SCOPE

In this thesis work, log files based prediction of failure types using Deep Learning models is presented. This system has achieved reasonably good recognition results of 87.0% using large amount of preprocessing and complicated feature extraction techniques. In all the experiments performed, this has been concluded that, by increasing the size of training data the efficiency of the system increases.

The most important aspect on which one may like to work in future is towards increasing the accuracy of the failure type prediction. This can possibly be achieved using a larger sized dataset consisting of a larger number of failure instances. Also, one can explore other variants of backpropagation learning algorithm like Adadelta, Adagrad, RMSprop and Adamax *etc*. The experiments can also be done on a larger level by taking different architectures of network models used. For example, one can increase the number of layers in the CNN, the filters used in CNN can also be experimented for improving the accuracy of prediction.

In addition to failure types prediction the proposed work might also be used to speed up the process of diagnosis: For example, if root causes can be recognized for each failure type in a reference dataset, identification of the most similar reference pattern would allow a first allocation of potential root causes for a failure that has occurred during runtime.

REFERENCES

[1] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, 1998.

[2] Nentawe Gurumdimma, Arshad Jhumka, Maria Liakata, Edward Chuah and James Browne, "Towards Increasing the Error Handling Time Window in Large-Scale Distributed Systems using Console and Resource Usage," in *Proceedings of the IEEE Trustcom/ BigDataSE/ ISPA, pp. 61-68*, 2015.

[3] Prerna Juneja, Divya Kundra and Ashish Sureka, "Anvaya: An Algorithm and Case-Study on Improving the Goodness of Software Process Models generated by Mining Event-Log Data in Issue Tracking Systems", in *Proceedings of the IEEE 40th Annual Conference on Computer Software and Applications* (*COMPSAC*), *)*, pp. 53-62, 2016.

[4] Christophe Bertero, Matthieu Roy, Carla Sauvanaud and Gilles Tredan, "Experience Report: Log Mining using Natural Language Processing and Application to Anomaly Detection", in *Proceedings of the IEEE 28th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 351-360, 2017.

[5] Errin W. Fulp, Glenn A. Fink and Jereme N. Haack, "Predicting Computer System Failures Using Support Vector Machine", in *Proceedings of the First USENIX conference on Analysis of system logs*, pp. 5-5, 2015.

[6] Min Du, Feifei Li, Guineng Zheng and Vivek Srikumar, "DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning", in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS '17)*, pp. 1285-1298, 2017.

[7] Felix Salfner and Steffen Tschirpke, "Error Log Processing for Accurate Failure Prediction", in *Proceedings of the First USENIX conference on Analysis of system logs*, pp. 4-4, 2012.

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

[9] Yamanishi, K., and Maruyama, "Dynamic syslog mining for network failure monitoring", in *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, pp. 499–508, 2005.

[10] Chunting Zhou, Chonglin Sun, Zhiyuan Liu and Francis C.M. Lau, "A C-LSTM Neural Network for Text Classification, Arxiv.org, 2015. [Online].
Available: https://arxiv.org/abs/1511.08630

[11] Y. Bengio, A. Courville, and P. Vincent. "Representation learning: A review and New perspectives.", in *Proceedings of the IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 38, no. 8, pp. 1798–1828, 2013.

[12] M Zeiler and Rob Fergus,"Visualizing and Understanding Convolutional Networks." in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 818-833, 2014.

[13] Sebastian Ruder, "An overview of gradient descent optimization algorithms", Arxiv.org, 2016. [Online].
Available: https://arxiv.org/abs/1609.04747

[14] Karpathy. CS231n Convolutional Neural Network for Visual Recognition [Online].
Available: http://cs231n.github.io/convolutional-networks/

[15] Manish Thapliyal, Recurrent Neural Networks Theory [Online].
Available:https://medium.com/mlrecipies/recurrent-neural-networks-theory-f81d59c2add7

[16] Shivam Bansal, A Comprehensive Guide to Understand and Implement Text Classification in Python [Online].
Available:https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/

[17] Tomer Levy, A Machine Learning Approach to Log Analytics [Online].
Available:https://logz.io/blog/machine-learning-log-analytics/

[18] Lak Lakshmanan, How to do text classification with CNNs, TensorFlow and word embedding [Online].

Available:https://towardsdatascience.com/how-to-do-text-classification-using-tensorflow-word-embeddings-and-cnn-edae13b3e575

[19] Jagreet Kaur Gill, Automatic Log Analysis using Deep Learning and AI for Microservices [Online].
Available:https://www.xenonstack.com/blog/data-science/log-analytics-deep-machine-learning-ai/

[20] Jason Brownlee, Best Practices for Document Classification with Deep Learning [Online].
Available:https://machinelearningmastery.com/best-practices-document-classification-deep-learning/

APPENDIX-A

PLAGIARISM REPORT

ORIGINALITY REPORT

11%	7%	5%	2%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.rok.informatik.hu-berlin.de Internet Source	2%
2	www.analyticsvidhya.com Internet Source	1%
3	www.xenonstack.com Internet Source	1%
4	pt.scribd.com Internet Source	1%
5	citeseerx.ist.psu.edu Internet Source	1%
6	"Neural Information Processing", Springer Nature, 2017 Publication	<1%
7	Min Du, Feifei Li, Guineng Zheng, Vivek Srikumar. "DeepLog", Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security - CCS '17, 2017 Publication	<1%

Prerna Juneja, Divya Kundra, Ashish Sureka.

8	"Anvaya: An Algorithm and Case-Study on Improving the Goodness of Software Process Models Generated by Mining Event-Log Data in Issue Tracking Systems", 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC), 2016 Publication	<1%
9	link.springer.com Internet Source	<1%
10	www.usenix.org Internet Source	<1%
11	Khanna, Nitin, Aravind K. Mikkilineni, George T. C. Chiu, Jan P. Allebach, Edward J. Delp, and Ping Wah Wong. "", Security Steganography and Watermarking of Multimedia Contents IX, 2007. Publication	<1%
12	Submitted to Singapore Institute of Technology Student Paper	<1%
13	dspace.uah.es Internet Source	<1%
14	Yang, Z.R.. "Searching for discrimination rules in protease proteolytic cleavage activity using genetic programming with a min-max scoring function", BioSystems, 200311 Publication	<1%

15	Mihandoost, S., M. Mazlaghani, M. Amirani, and A. Mihandoost. "Automatic feature extraction using generalised autoregressive conditional heteroscedasticity model: an application to electroencephalogram classification", IET Signal Processing, 2012. Publication	<1 %
16	www.cpt.univ-mrs.fr Internet Source	<1 %
17	"EVALUATION OF ANTI-BACTERIAL ACTIVITY, GC/MS ANALYSIS AND GENOTOXIC POTENTIAL OF CARUM COPTICUM ESSE", Journal of Animal and Plant Sciences, June 30 2016 Issue Publication	<1 %
18	"Proceedings of the Twelfth International Conference on Management Science and Engineering Management", Springer Nature, 2019 Publication	<1 %
19	Submitted to Queensland University of Technology Student Paper	<1 %
20	"Classification in BioApps", Springer Nature, 2018 Publication	<1 %

21	"Intelligent Computing Theories and Application", Springer Nature, 2017 Publication	<1%
22	"MultiMedia Modeling", Springer Nature, 2017 Publication	<1%
23	Submitted to Universiti Malaysia Perlis Student Paper	<1%
24	Submitted to University of Florida Student Paper	<1%
25	Levine, E.R.. "Classifying soil structure using neural networks", Ecological Modelling, 19961121 Publication	<1%
26	Submitted to University of Missouri, Kansas City Student Paper	<1%
27	Submitted to Asia Pacific University College of Technology and Innovation (UCTI) Student Paper	<1%
28	Zhao, W.. "Comparison of two cluster analysis methods using single particle mass spectra", Atmospheric Environment, 200802 Publication	<1%
29	Pawel Cyrta, Tomasz Trzciński, Wojciech Stokowiec. "Chapter 10 Speaker Diarization	<1%

	Using Deep Recurrent Convolutional Neural Networks for Speaker Embeddings", Springer Nature, 2018 Publication	
30	Jiuwen Cao, Zhiping Lin, Guang-bin Huang. "Composite function wavelet neural networks with extreme learning machine", Neurocomputing, 2010 Publication	<1%
31	Submitted to Mansoura University Student Paper	<1%
32	Christophe Bertero, Matthieu Roy, Carla Sauvanaud, Gilles Tredan. "Experience Report: Log Mining Using Natural Language Processing and Application to Anomaly Detection", 2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE), 2017 Publication	<1%
33	dyuthi.cusat.ac.in Internet Source	<1%
34	Damla Arifoglu, Abdelhamid Bouchachia. "Activity Recognition and Abnormal Behaviour Detection with Recurrent Neural Networks", Procedia Computer Science, 2017 Publication	<1%
35	Submitted to University of Queensland Student Paper	<1%

36	repository.upi.edu Internet Source	<1%
37	Submitted to University of Modena and Reggio Emilia Student Paper	<1%

Exclude quotes	On	Exclude matches	< 10 words
Exclude bibliography	On		