# Chapter 5
# A Novel Approach of Software Fault Prediction Using Deep Learning Technique


**Debolina Ghosh and Jagannath Singh**

**Abstract** Now-a-days, failure of the software is unavoidable due to increasing size and complexity of software. So, fault finding is necessary for removing the software faults. Spectrum-based fault localization is most popular technique to find the faulty statements of a given program. Still, there are some limitations also. In case of large software, it is very hard and time taking to test all possible scenarios via traditional approach. The machine learning model is an interesting approach for solving this. Recently, deep learning is widely used for improving the fault finding techniques. Deep learning models are based on the architecture of neural network. The neural network architectures based on input layer, hidden layer(s) and output layer. Convolution Neural Network (CNN) is a well-known architecture for deep learning. The network is trained with large amounts of data and neural network architectures that learn the features directly from the information. So, there is no need of manual feature extraction. This technique can capable of finding the suspicious score of each program statement. Using this technique, we can collect the large amount of data from the test cases and extract the important features. As pooling layer of CNN model reduces the input size and complexity of the model, so it speeds up the training process. This framework can also be able to calculate the suspicious score of each statement and accordingly assign the rank.

**Keywords** Testing · Debugging · Software fault prediction · Spectrum-based debugging · Deep learning · Convolution neural network

D. Ghosh (✉) · J. Singh
Kalinga Institute of Industrial Technology, Bhubaneswar, India
e-mail: jagannath.singhfcs@kiit.ac.in

© Springer Nature Switzerland AG 2020

S. C. Satapathy et al., *Automated Software Engineering: A Deep Learning-Based Approach*, Learning and Analytics in Intelligent Systems 8,
https://doi.org/10.1007/978-3-030-38006-9_5

## 5.1 Introduction

A good quality software is always in high demand and proper testing will ensure the quality of a product. Now-a-days, LOCs and size of software are increasing and hence testing needs more time and effort. Software testing is done in two phases: testing and debugging [5]. In the first phase, the test cases for program under test are generated and observed whether the program behaves as expected or not. Many automated software testing techniques are present to generate test cases and to check the program behavior. In the next phase, the root cause of the error is discovered and correct it. The debugging phase needs much more time and effort than testing. Few research work has been carried out on automated debugging tools [8, 22, 25]. Nonetheless, automated debugging tools are needed for better software debugging.

We are presenting some recent debugging techniques in this section. In our literature, we find spectrum-based debugging techniques [6] can be used for developing automated debugging tool. For single and multi-fault programs, genetic algorithm (GA) gives satisfactory results [32]. The genetic algorithm is an evolutionary process which solves the optimization problem based on natural selection. In modern society, machine learning algorithms are widely used in different aspects, from web searches to content filtering, different e-commerce websites, smartphone cameras, driverless car and many more [19]. Deep learning or deep structured learning [18] is a part of machine learning technology. Machine learning is a subset of artificial intelligence. Deep learning is a class of machine learning algorithms for feature extraction and transformation for solving non-linear complex problems. Higher level features are obtained from the lower level features and make a hierarchical structure. Basically, machine learning is a shallow learning whereas deep learning is a hierarchical learning with abstraction. Now-a-days, deep learning techniques are rapidly developing to make the system more robust and accurate. It has already been applied on different research domain like image processing, text summarization, pattern recognition etc. Thus, deep neural network with multiple hidden layers can be applied for fault localization technique also. In this chapter, we present Convolution Neural Network (ConvNet or CNN) for multi-fault localizations. CNN is a class of deep neural network and a regularized version of multi-layer perceptron (MLP) which refers to fully connected networks. CNN has the fewer connections and parameters in compared to MLP. Firstly, the CNN model is trained with test cases and then, by checking the trained model with virtual test suite, it calculates the suspicious score of each program statement.

The remaining section of the paper is organized as follows: Sect. 5.2 contains some details of the related work. Section 5.3 explains several basic concepts of deep learning and various current deep learning architectures. Section 5.4 describes the proposed CNN structure for multi-fault localization and Sect. 5.5 ends the paper.

## 5.2 Related Work

In this section, some of the closely related research works in the field of software debugging and fault localization is presented.

Ribeiro et al. [21] presented a spectrum based fault localization open source tool Jaguar (JAva coveraGe faUlt locAlization Ranking) for Java programs. It uses the fault localization technique of the data-flow and includes the visualization of lists of suspect statements that are more likely to be faulty. Zhang et al. [28] presented PRFL, follows PageRank algorithm for boosting spectrum-based fault localization. The tool analyzed the importance of each test and accordingly assign the rank to each methods in a given program.

Campos et al. [3] presented GZoltar, an automatic fault localization tool based on spectrum-based fault localization (SBFL). The tool is used as an Eclipse plug-in to reduce and prioritize the test suite. The tool provides a graphical view of faulty and non-faulty statements. Wong et al. [26] introduced a D-Star technique of fault localization which measures the suspicious score and identifies the faulty statement.

Chakraborty et al. [4] developed a spectrum-based debugging tool to find the faulty statements using different statistical formulas. Zheng et al. [32] developed a multi-fault localization tool (FSMFL) based on spectrum-based fault localization and genetic algorithm. FSMFL is applied on different open-source benchmark programs and the result shows that the tool performs better than existing fault localization tools for locating first fault of multi-fault program.

Eniser et al. [7] presented DeepFault, Deep Neural Network (DNN)-based white box analysis technique which detects suspicious neurons and synthesizes a new input which improves suspicious neuron activation values. The researchers used this technique for MNIST and CIFAR-10 datasets and showed that the suspect neurons can be identified by DeepFault.

Zheng et al. [31] proposed a deep neural network based fault localization technique for single-fault programs. They have conducted the experiment on siemens suite and space program and it gives the satisfactory result for locating the fault.

Zhang et al. [30] have presented slicing with deep learning-based technique of fault localization that can measure suspicious score of a faulty statement. To extract the contextual information, dynamic backward slicing is then applied. They have compared their results with another fault localization technique, D-Star.

Zhang et al. [29] have presented a single-fault localization technique based on Convolution Neural Network (CNN). It uses virtual test suite to evaluate the suspicious score of each program statement by testing the test cases. The experimental result shows that the proposed approach improves the effectiveness of fault localization technique for single-fault programs.

## 5.3 Basic Concepts

We present some relevant concepts in this section that will need to understand our work.

### *5.3.1 Deep Learning*

Artificial intelligence [16] is a machine which has the capability to imitate the behaviour of human brain. Machine learning is a branch of artificial intelligence that makes it possible for a machine to learn. But machine learning algorithms are not able to handle high dimensional data where input and output data is large. In this scenario, machine learning algorithms are not perform well whereas deep learning algorithms perform satisfactorily. Deep Learning [11, 18] is a part of machine learning algorithms that uses multiple layers to extract the features from the given original input data. It is a new state-of-the-art for artificial intelligence. Deep learning algorithms run through a different states of neural network algorithms. Neural network is a set of neurons that work in the same manner as individual neurons of the human brain. Deep learning models operate on huge amount of data through multiple layers and can able to solve complex problems. Different architectures of deep learning, such as deep neural networks, recurrent neural networks, and convolution neural networks have applied various fields of computer science. The relation between artificial intelligence, machine learning and deep learning is shown in Fig. 5.1.

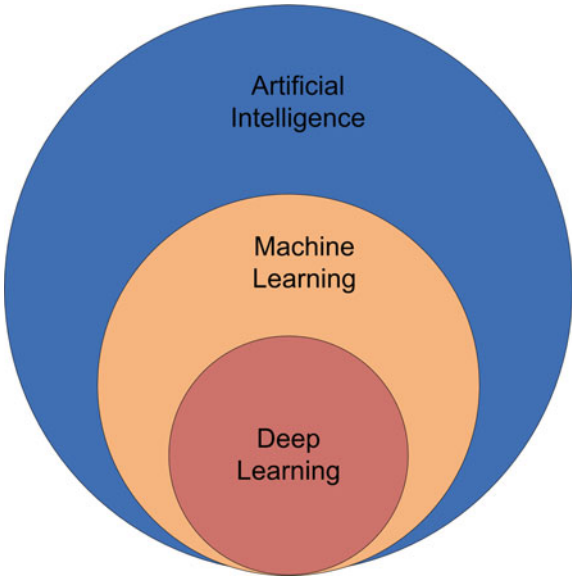


**Fig. 5.1** Relation of deep learning, machine learning and AI

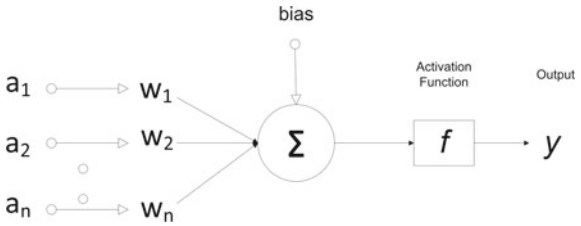


**Fig. 5.2** Structure of neuron

### *5.3.2 Basic Deep Learning Terminologies*

There are different deep learning terminologies which are widely used in neural network models.

**Neuron**: In deep learning, neuron is just like the neuron of human brain. When it receives any information, it process it and generate the output. Similarly in neural network, the neuron takes the input, process it and send the output to the next neuron, as shown in Fig. 5.2.

**Weights**: When the input value enters into neuron, it the multiplied by weight e.g. a neuron having 3 inputs and each input is multiplied by its corresponding weight. Initially, weight is randomly generated. During the training process weight can be updated. An input a after passing through a node, it becomes a * w, where w is the assigned weight for input a in Fig. 5.2.

**Bias**: Bias is a linear component added to the input after multiplied by weight. Bias is generally used to change the range of the weight multiplied by input. After bias is added, the output value becomes a * w + bias.

**MLP (Multi Layer Perceptron)**: A perceptron is a linear binary classifier. A single neuron is not able to solve complex problems. Therefore, multiple neurons are required in specific multiple layers, i.e. input layer, hidden layer, and output layer. Each layer consists of multiple neurons. All the neurons in a layer are linked with all the neurons in the next layer. Multi layer perceptron is also known as fully connected network [9, 20].

**Activation function**: Different activation functions [1, 23] are used in neural networks. Activation function is necessary to incorporate non-linearity into a neuron's output. By introducing non-linearity, the activation function makes the model more capable to learn and perform the complex tasks. Without activation function the model works as a simple linear regression model which doesn't perform satisfactory most of the times. Each neuron works on the basis of weight, bias and the activation function. The weights and biases are changed in each neuron on the basis of the error at the output. This process is called Back-Propagation. So, during back-propagation strategy we need activation function to optimize the weight or to apply another optimization technique to reduce the error.

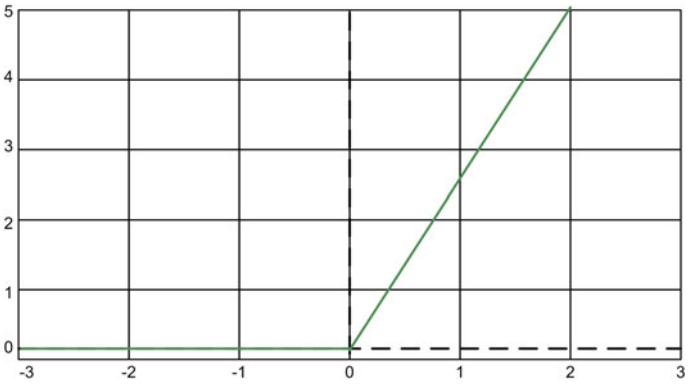


**Fig. 5.3** ReLu activation function

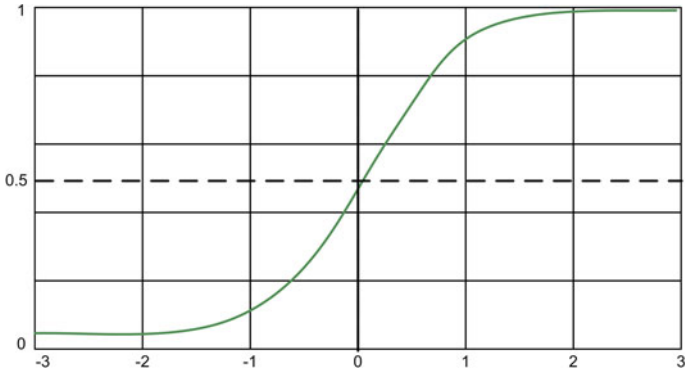


**Fig. 5.4** Sigmoid activation function

Different activation functions-

1. **ReLu function**: ReLu is a Rectified linear units which is mostly used activation function with the form of $R(x) = \max(0, x)$ i.e if $x < 0$, $R(x) = 0$ and if $x \geq 0$, $R(x) = x$. ReLu is very simple, efficient and more faster than any other activation functions. It avoids the vanishing gradient problem. But the drawback of Relu is, it can be only used within the hidden layers of neural network model. The role to activate ReLu is shown in Fig. 5.3.
2. **Sigmoid function**: This activation function is in the form of $f(X) = 1/1 + \exp(-x)$ with the range from 0 to 1. The curve is S-shaped, as shown in Fig. 5.4. The output value lies between 0 and 1 and it has slow convergence.
3. **Softmax function**: Softmax activation function is almost similar to Sigmoid function. It is used when we try to solve the classification problem. It is used in the output layer of the classifier.

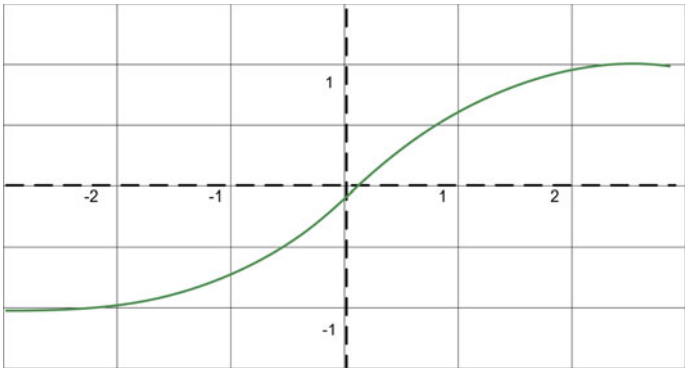


**Fig. 5.5** Tanh activation function

4. **Tanh function**: Tanh function is Tangent Hyperbolic function. It can be derived from sigmoid function. It is in the form of $f(x) = 2/(1 + e^{-2x}) - 1$ or $\tanh(x) = 2 * sigmoid(2x) - 1$. The range of tanh function lies between $-1$ to 1, as shown in Fig. 5.5. As the optimization is easy in this function, so it is preferred over Sigmoid function. But tanh function is suffered from vanishing gradient problem.

**Forward propagation**: Forward propagation [10] is the movements of inputs to the output layers through hidden layers. There is no backward movements. The information travers only in forward direction. The input layer provides the hidden layer data and the output is generated.

**Cost function**: Each neural network tries to predict the output after each iteration. If the resultant output is as close as the actual output, then the error is minimal. This accuracy is measured by the cost or loss function. So, when we train the network our main aim is to minimize the cost or loss function.

**Back propagation**: Neural network assigns weight for each inputs and assign bias value to the nodes. After one iteration, from the generated output we can find-out the error. Then, to change the weight value, this error is fed back to the network along with its cost function. Then each input weight is then updated to minimize the error. This process is called back-propagation [13].

**Batches**: During the training process, we send the input data into small chunks instead of sending whole input data at a time. This data chunks is called batches. Sending the data into batches makes the model more generalized and also minimizes the error.

**Epochs**: Epochs is defined by a single iteration of all batches in both backward and forward propagation. Epochs is used during the training process of the network. If the number of epochs is high, it will take the longer time for the network to converge.

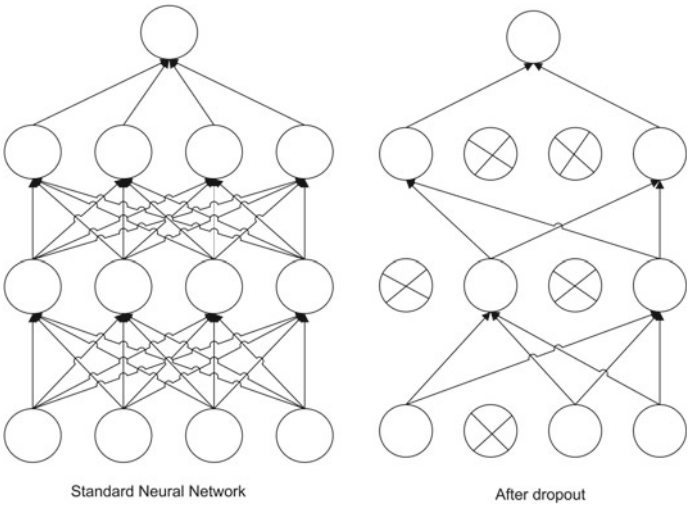


**Fig. 5.6** Dropout

**Dropout**: Dropout [14] is a regularization process to prevent overfitting problem in the network. Some of the neurons are dropped in the hidden layer during the training process. After dropping out some of the neurons generate the output, as shown in Fig. 5.6.

**Learning rate**: Learning rate is count that the amount of weights are updated during each iteration. It determines at what extent the new information overwrites the old information. The learning rate is usually between 0 and 1. If the learning speed is very small it takes forever for the network to converge and if the rate is too high it may skip the solution.

**Vanishing gradient problem**: Vanishing gradient problem [15] arises in activation function where the gradient value is very small. During the training process, the gradient value is multiplied with weight and the value becomes negligible in the next iterations. So the network forgets the long range dependency. The vanishing gradient problem generally occurs in Recurrent neural network where long range dependency is very important. This problem can be avoided by using another activation function ReLu which doesnot have small gradient value.

### *5.3.3 Deep Learning Models*

Deep learning architectures are based on artificial neural network. A computer model can perform a specific task from an object like image, voice, speech, pattern etc. To train a model we send a large amount of input data and it extracts the features through

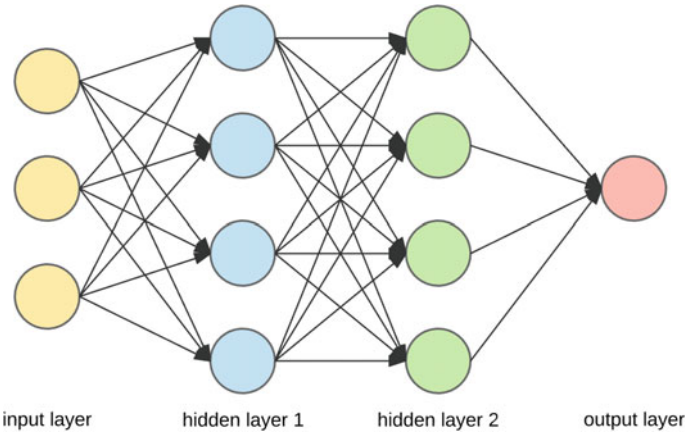


**Fig. 5.7** Structure of DNN

several layers from the dataset. There are different types of deep learning models which are used in different fields of computer science.

**Deep Neural Network (DNN)**: Neural network is a special technology which simulate the activity of human brain. Deep neural network [24] is a multi-layered neural network with a certain level of complexity. DNNs have one layer of input and output and several hidden layers, shown in Fig. 5.7. Every layer trains on a set of characteristics based on the previous layer's output. Each layer accumulated features from the previous layer and recombined them. This process is known as the hierarchy of features, which enhances complexity and abstraction. This type of neural network is used with unlabeled and unstructured data.

**Recurrent Neural Network (RNN)**: Recurrent Neural Network [27] is one kind of neural network where output of one layer send to next layer and output of that layer again send back to that layer, as shown in Fig. 5.8. RNN maintains a memory which stores all the information of previous step. Thus, RNN is also known as long short-term memory networks (LSTMNs). It performs same task in each hidden layer, so it reduces the complexity of parameters. RNNs are used for time series problem, speech recognition and driver-less car etc.

**Convolution Neural Network (CNN)**: Convolution Neural Network [2] is a multi-layer perceptron that reduces the processing requirements. CNN is like a flash-light like structure. CNN layers consist of an input layer, an output layer and a hidden layer comprising a convolution layer, a pooling layer, a fully connected layer, as shown in Fig. 5.9. A given neuron has the same number of input neurons and the same weight and biases. Pooling layer minimizes the input neuron size and boots the rate of learning. CNNs are easier to train and have fewer parameters in compared to other deep learning architectures. CNNs are widely used in image processing, recommender system, video recognition and natural language processing.

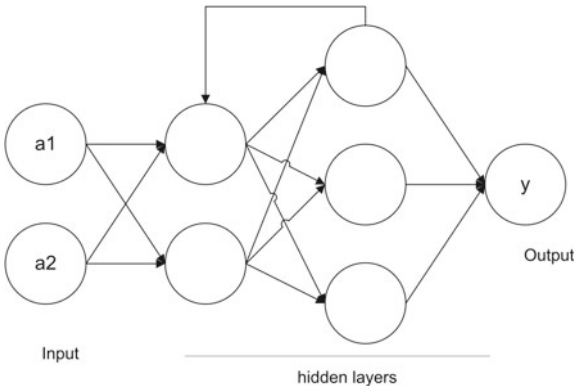


**Fig. 5.8** Structure of RNN

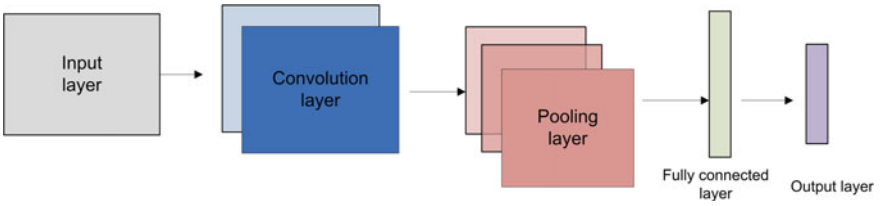


**Fig. 5.9** Structure of CNN

## 5.4 Fault Localization Using CNN

In fault localization, there is always a greater number of program statements and test cases. As an example, if a program has 100 statements and if 100 statements works as input of deep learning network with same size of hidden layers then it will be 100 * 100 = 10,000. It makes the system more complex. So we need to reduce the parameters and the complexity of the system.

Multiple hidden layers or too many inputs make the network complex, which is called **overfitting problem**. The solution of over-fitting problem is:

- Split the data into multiple sets e.g. Training set, Test set, Cross-validation set etc.
- Regularization of model i.e divide the neurons by weight.
- Max norm constraints i.e. add size limits according to weights and biases.
- Dropout means randomly off some neurons to solve the overfitting problem.

### 5.4.1 Framework Overview

In this section, we present the architecture of convolution neural network for multi-fault localization. CNN [12] is a deep learning network which reduces the number of parameters in each step. CNN's architecture consists of five steps: the layer of input, the layer of convolution, the layer of pooling, the layer of output and fully connected layer. Using a filter, the convolution layer slides the filter over an input layer. The size of the input is presented in a matrix form. It performs the dot product between the selected input size and filter size. So the convolution layer consists of a set of independent filters. Next, pooling layer is used to reduce the size and complexity of the network's input parameters. It scaling down the input and keeps only the important features for the next layer. After several convolution layers and pooling layers, fully connected layer is connected to all previous layer activation.

For fault localization using CNN, first we initiate the convolution with training dataset. Each statement is performed by M test cases for a given program P of N statements, as shown in Table 5.1. The error vector $e_1, e_2 \ldots e_n$ represent the test case execution results either 1 (if passed) or 0 (if failed). The statement $a_{ij} = 1$ means that ith test case executes the statement j. The statements execution information with the test case results work as input of the convolution layer. The training process will train the model. Then we create a set of virtual test cases that will act as test data. Table 5.2 shows the matrix of virtual test cases. We create the virtual test cases that are equal to the number of program statements. Each test case will execute by only one statement. If $a_i = 1$ i.e statement i is executed by test case $t_i$. When the virtual test case matrix is inserted into the trained CNN model , it calculates the suspicious score of each program statements. The suspicious score is the approximation of the outcomes of the virtual test case execution covered by one statement. The larger value denotes the more suspicious statement.

**Table 5.1** The coverage and results of execution

| N statements | | | | Errors |
|---|---|---|---|---|
| $a_{11}$ | $a_{12}$ | … | $a_{1N}$ | $e_1$ |
| $a_{21}$ | $a_{22}$ | … | $a_{2N}$ | $e_2$ |
| . | . | … | . | . |
| . | . | | . | . |
| $a_{M1}$ | $a_{M2}$ | … | $a_{MN}$ | $e_N$ |

**Table 5.2** Virtual testcase

| | $a_1$ | $a_2$ | … | $a_N$ |
|---|---|---|---|---|
| t1 | 1 | 0 | … | 0 |
| t1 | 0 | 1 | … | 0 |
| . | . | … | . | . |
| . | . | | . | . |
| tN | 0 | 0 | … | 1 |

Figure 5.10 describes the procedure of fault localization using CNN. The statement coverage data and the outcome of each test case are sent to the model in the input layer. This is the training process. This data will train the model by following the each stage. After the model is properly trained, the test data (i.e. virtual test cases) will sent to the model to generate the suspicious score of each statements.

The convolution model [17] consists of different layers.

1. **Convolution Layer**: The first layer in CNN is the Convolution layer. It extracts the features from the input. It takes the matrix of input, e.g. h * w * d, where h is the height, w is the width, and d is the input matrix dimension. Each input will go through a series of filters or kernels. It performs the dot product and produce the Featured Map. The filter moves from left to right until it completes the width and then goes to down. This process is keep on iterating until the input matrix is completed. In Fig. 5.11, the size of input matrix is 5 * 5 and it convolutes with filter matrix of size 3 * 3. The process performs dot operation and generate the feature map of size 3 * 3.
2. **Pooling Layer**: When the input size is increasing, pooling layer minimizes the parameters. It reduces the complexity of the software by decreasing the size of the matrix. It is used to extract the dominant features from the input. Two types of pooling layers are available, Max-Pooling and Average-Pooling.
   **Max pooling** takes the maximum value from the kernel or filter-covered matrix, as shown in Fig. 5.12.
   **Average pooling** takes average value from the portion covered by the filter or kernel. In our study we consider max pooling as it performs better than average pooling.
3. **Fully Connected Layer**: Fully connected layer is a multi layer perceptron where we flattened the matrix into vector and feed it into a fully connected network. Back-propagation is applied in every iteration. Feature map matrix is converted into vectors and feed to the fully connected networks, as shown in Fig. 5.13.
4. **Output Layer**: The last layer that can classify the data is the output layer. After applying the activation functions the model will be properly trained and can classify the outputs. In our study, we apply sigmoid activation function as we need to generate the score of each statement which lies between 0 and 1. Here, the output layer provides the suspicious score of each program statements.

### *5.4.2 Experimental Setup and Example Program*

To perform this experiment, we need a system with 3 GHz Intel(R) i5 CPU , 32 GB primary memory and python 3.5 or above.

In Table 5.3, we have taken an example program P of 26 lines. Total executable statements of the program P is 17. In the example program, there are two faulty statements 14 and 17. Last row represents the test case results i.e pass or fail. If the

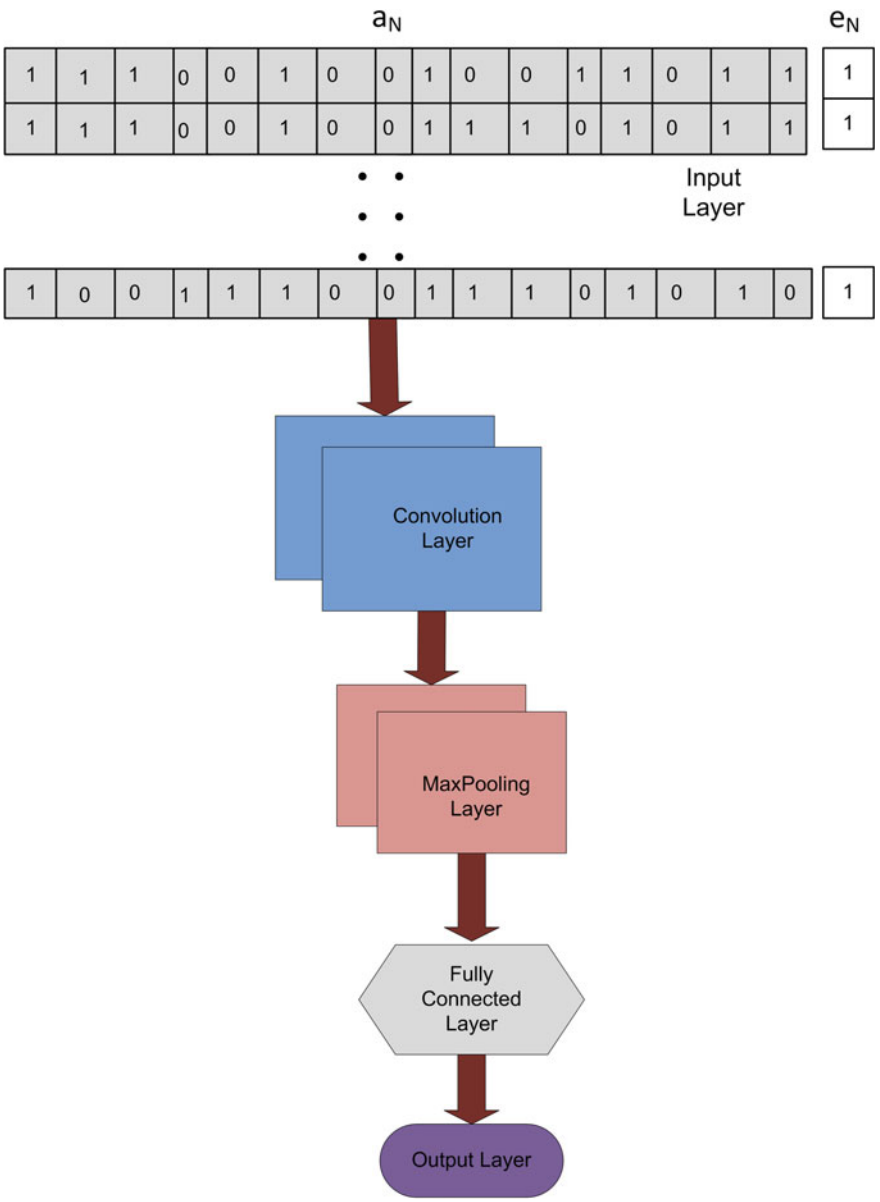


**Fig. 5.10** Framework of CNN

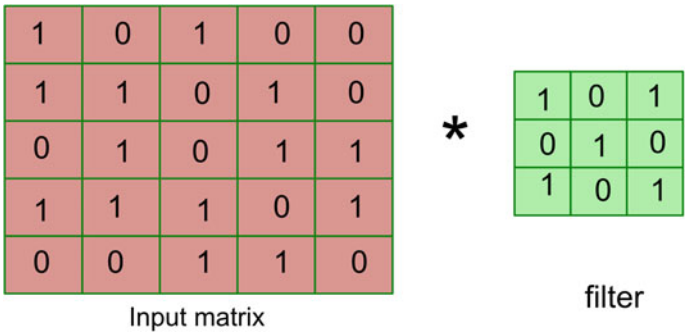


**Fig. 5.11** Convoluting 5 * 5 matrix with 3 * 3 filter matrix

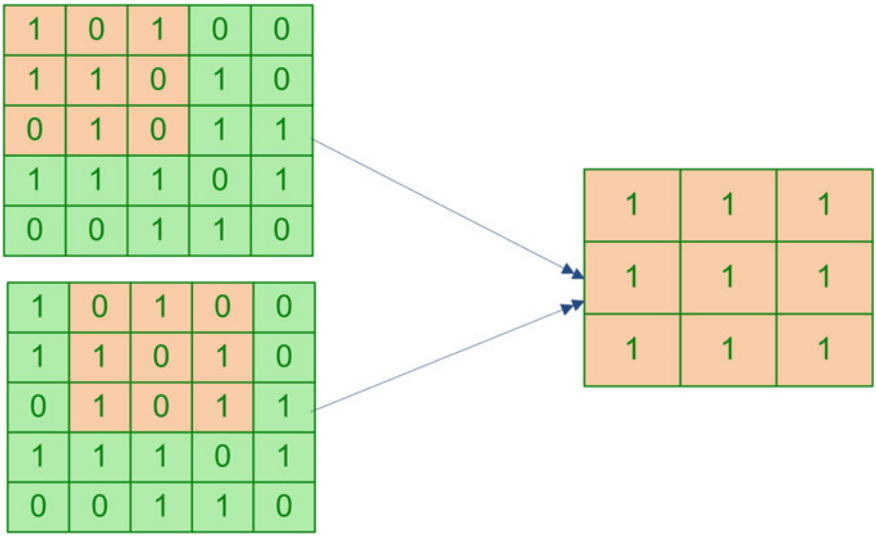


**Fig. 5.12** Max pooling

result is pass, we will consider it as 1 otherwise 0. Among 10 test cases 2 test cases are failed T3 and T4.

First, We will create CNN model with layer of input, layer of convolution, layer of pooling followed by fully connected layer and layer of output. The output node consists of a single node and sigmoid function will be used as an activation function. In the next stage, we will train the model with statement coverage data and with test case results as shown in Table 5.1. Firstly, we will send the training data accordingly the filter size which we have set during the construction of CNN model. If the filter size is 3, we will send the statement coverage of T1, T2, T3 and the execution results (1 1 0). In the next iteration, we will send the statement coverage of T4, T5, T6 and the execution results (0, 1, 1) and so on. After the model is trained, we will send the

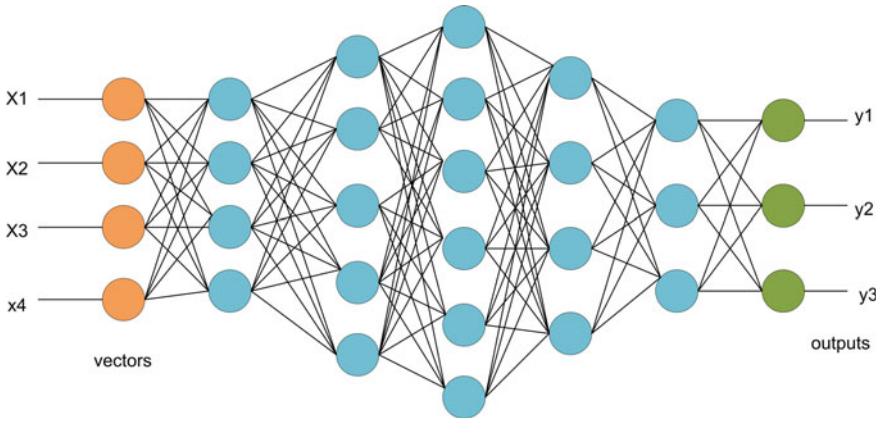


**Fig. 5.13** Fully connected layer

test data or virtual test suite for fault localization. The virtual test suite consists of 17 test cases in which only one statement executes each test case. After executing this step, the model will generate the suspicious score of each program statements. The statements with higher suspicious score will have the probability of being faulty.

### *5.4.3 Procedure of Fault Localization Using CNN*

In this subsection, we discuss the details procedure of multi-fault localization using different layers of CNN. The steps are shown in Algorithm 1.

1. First, we read the input dataset from the given location. In our study, the input dataset is the statement coverage information with its test case results. In our example program in Table 5.3, the executable statements are 17 and number of test cases are 6. So, coverage matrix must be in dimension $6 \times 17$.
2. In the next stage, we need to divide the dataset with input data and output data. Again input dataset and output dataset are split with training set and test set to remove the complexity problem or overfitting problem.
3. Then, we create the Convolution model.
4. In this step, multiple Convolution layer and Pooling layer are added depending upon the input dataset.
5. Fully connected layers connect one neuron in one layer to each neuron in another layer. It is a traditional Multi-layer perceptron (MLP).
6. Dense layer is the output layer which applies the activation function to the previous layer values. Once the activation function is applied (e.g. Relu, Sigmoid), the output layers generates the score between 0 and 1 for each statements. The statements with higher score denotes more suspicious statements.

**Table 5.3** An example program with statements

| Find second number among four | T1 | T2 | T3 | T4 | T5 | T6 | T7 | T8 | T9 | T10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1. int find_second(int a, int b, int c, int d){ | | | | | | | | | | |
| 2. int max,min,temp; | | | | | | | | | | |
| 3. if(a<=b){ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 4. max=b; | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5. min=a; | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6. } else { | | | | | | | | | | |
| 7. max=a; | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 8. min=b; | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 9. } | | | | | | | | | | |
| 10. if(c > max){ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 11. temp=max; | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 12. max=c; | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 13. }else if(c < min){ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 14. /*bug, right is temp =min * temp=b; | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 |
| 15. min=c; | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 |
| 16. }else { | | | | | | | | | | |
| 17. /* bug , right is temp =c */ temp=b; | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |
| 18. } | | | | | | | | | | |
| 19. if d < max && d > temp){ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 20. return d; | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 21. { else if(d > max) { | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 22. return max; | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 23. { else { | | | | | | | | | | |
| 24. return temp; | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 25. } | | | | | | | | | | |
| 26. } | | | | | | | | | | |
| Test Case Result | P | P | F | F | P | P | P | P | P | P |

7. In this stage, the model is complied and trained with the input dataset. We can choose the batch size and epochs during the training process. Batch size is dependent on the size of training data. Epochs is normally set to 1 i.e for each batch the model will be trained only once.
8. In this step, the trained model will predict the output of test data and compare with the actual output. If there is error, the model fed back to the neuron , update the weights and again train the model. This back-propagation process continues until the error is minimized.

**Algorithm 1** CNN(X,Y, ConV2D, MaxPool, Dense)

**INPUT:** Training dataset, Test dataset
**OUTPUT:** Suspicious score of program statements
**procedure** CONVOLUTION NEURAL NETWORK(Input datasets)
dataset=read_dataset() ▷ read the input dataset
*X*=divide_dataset()
*Y*=divide_dataset()
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size) ▷ Didive the datasets to solve overfitting problem
model = sequential() ▷ create the model
**for** model.add(ConV2D(input_size, filter_size, activation_function, inputShape))
model.add(Maxpooling(size)) ▷ Ctreate pooling layer **do**
**end for**
model.add(falttern()) ▷ Fully connected layer
model.add(Dense(size, activation_function)) ▷ Output layer
compile.model() ▷ compile the created model
model.fit(X_train, Y_train, batch_size, epchos) ▷ train the model
model.predict(X_test) ▷ predict the model
model.compare(Y_test) ▷ compare with actual output
return suspicious score of each statements
**end procedure**

## 5.5 Conclusion and Future Work

Machine learning is an artificial intelligence technology that produces a machine with knowledge that can learn and work. Machine learning algorithms access the data or information, learn from it and then predict the future output. But when the data is huge, the machine learning algorithms are not sufficient to handle those data and their performance is not satisfactory. So the concept of deep learning comes. Deep learning is a part of artificial neural network-based machine learning technique. Deep learning algorithms are motivated by the function and structure of human brain. Now-a-days, deep learning is a very emerging field. Different deep learning models are used in different fields of computer science. In this chapter, using the Convolution Neural Network, we propose a multi-fault localization technique. First, we need to construct the model and send the statement coverage data and error vectors as input to the convolution layer. After the model is trained, we need to send the test data or the virtual test cases for fault localization. The generated output will give the suspiciousness score of each statements. For multi-fault localization, we consider CNN because it reduces the parameters and complexity of the software, so it speeds up the training process.

In future, we plan to implement CNN on well known multi-fault benchmark programs. We have also planned to implement Recurrent Neural Network (RNN) to localize the multiple faults.

## References


1. F. Agostinelli, M. Hoffman, P. Sadowski, P. Baldi, Learning activation functions to improve deep neural networks. ArXiv preprint arXiv:1412.6830 (2014)
2. D. Britz, Understanding convolutional neural networks for NLP (2015). http://www.wildml.com/2015/11/understanding-convolutional-neuralnetworks-for-nlp/. Visited on 11 July 2015
3. J. Campos, A. Riboira, A. Perez, R. Abreu, Gzoltar: an eclipse plug-in for testing and debugging. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering* (ACM, 2012), pp. 378–381
4. S. Chakraborty, Y. Li, M. Irvine, R. Saha, B. Ray, Entropy guided spectrum based bug localization using statistical language model. ArXiv preprint arXiv:1802.06947 (2018)
5. N. Chauhan, *Software Testing: Principles and Practices* (Oxford University Press, 2010)
6. H.A. de Souza, M.L. Chaim, F. Kon, Spectrum-based software fault localization: a survey of techniques, advances, and challenges. ArXiv preprint arXiv:1607.04347 (2016)
7. H.F. Eniser, S. Gerasimou, A. Sen, Deepfault: fault localization for deep neural networks. ArXiv preprint arXiv:1902.05974 (2019)
8. P. Fritzson, N. Shahmehri, M. Kamkar, T. Gyimothy, Generalized algorithmic debugging and testing. ACM Lett. Programm. Lang. Syst. (LOPLAS) **1**(4), 303–322 (1992)
9. M.W. Gardner, S. Dorling, Artificial neural networks (the multilayer perceptron) a review of applications in the atmospheric sciences. Atmos. Environ. **32**(14–15), 2627–2636 (1998)
10. X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (2010), pp. 249–256
11. I. Goodfellow, Y. Bengio, A. Courville, *Deep Learning* (MIT Press, 2016)
12. X. Guo, L. Chen, C. Shen, Hierarchical adaptive deep convolution neural network and its application to bearing fault diagnosis. Measurement **93**, 490–502 (2016)
13. R. Hecht-Nielsen, Theory of the backpropagation neural network, in *Neural Networks for Perception* (Elsevier, 1992), pp. 65–93
14. G.E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, R.R. Salakhutdinov, Improving neural networks by preventing co-adaptation of feature detectors. ArXiv preprint arXiv:1207.0580 (2012)
15. S. Hochreiter, The vanishing gradient problem during learning recurrent neural nets and problem solutions. Int. J. Uncertainty Fuzziness Knowl.-Based Syst. **6**(02), 107–116 (1998)
16. A.K. Jain, J. Mao, K. Mohiuddin, Artificial neural networks: a tutorial. Computer **3**, 31–44 (1996)
17. A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, in *Advances in Neural Information Processing Systems* (2012), pp. 1097–1105
18. Y. LeCun, Y. Bengio, G. Hinton, Deep learning. Nature **521**(7553), 436 (2015)
19. B.K. Natarajan, *Machine Learning: A Theoretical Approach* (Elsevier, 2014)
20. S.K. Pal, S. Mitra, Multilayer perceptron, fuzzy sets, and classification. IEEE Trans. Neural Netw. **3**(5), 683–697 (1992)
21. H.L. Ribeiro, H.A. de Souza, R.P.A. de Araujo, M.L. Chaim, F. Kon, Jaguar: a spectrum-based fault localization tool for real-world software, in *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)* (IEEE, 2018), pp. 404–409
22. J. Singh, P.M. Khilar, D.P. Mohapatra, Code refactoring using slice-based cohesion metrics and aspect-oriented programming. Int. J. Bus. Inf. Syst. **27**(1), 45–68 (2018)
23. D.F. Specht, Probabilistic neural networks. Neural Netw. **3**(1), 109–118 (1990)
24. C. Szegedy, A. Toshev, D. Erhan, Deep neural networks for object detection, in *Advances in Neural Information Processing Systems* (2013), pp. 2553–2561
25. G.M. Wambugu, K. Mwiti, Automatic debugging approaches: a literature review (2017)
26. W.E. Wong, V. Debroy, R. Gao, Y. Li, The dstar method for effective software fault localization. IEEE Trans. Reliab. **63**(1), 290–308 (2014)

27. W. Zaremba, I. Sutskever, O. Vinyals, Recurrent neural network regularization. ArXiv preprint arXiv:1409.2329 (2014)
28. M. Zhang, X. Li, L. Zhang, S. Khurshid, Boosting spectrum-based fault localization using pagerank, in *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis* (ACM, 2017), pp. 261–272
29. Z. Zhang, Y. Lei, X. Mao, P. Li, in CNN-FL: an effective approach for localizing faults using convolutional neural networks, in *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)* (IEEE, 2019), pp. 445–455
30. Z. Zhang, Y. Lei, Q. Tan, X. Mao, P. Zeng, X. Chang, Deep learning-based fault localization with contextual information. IEICE Trans. Inf. Syst. **100**(12), 3027–3031 (2017b)
31. W. Zheng, D. Hu, J. Wang, Fault localization analysis based on deep neural network, in *Mathematical Problems in Engineering* (2016)
32. Y. Zheng, Z. Wang, X. Fan, X. Chen, Z. Yang, Localizing multiple software faults based on evolution algorithm. J. Syst. Softw. **139**, 107–123 (2018)