# MODEL

## 4.1 Dataset

In this section, the data sets used to train, validate and test the Convolutional Neural Network model will be introduced.

The data sets are from two sources, one is the code data extracted from different J2EE frameworks and interfaces; the other is from BigCloneBench dataset which is a large dataset for many levels java code clones[38].

As a conclusion, there are 5371 pairs of similar programs are used as positive data sets in training process; 5020 pairs of not similar programs are used as negative data sets in training process; 360 pairs of similar programs are used as positive data sets in testing process and 213 pairs of not similar programs are used as negative data sets in testing process.

4.1.1 J2EE

Java Enterprise Edition, known as Java 2 Platform, Enterprise Edition (J2EE). J2EE is a collection of functional specific projects which extends Java SE 8. It includes enterprises such as Web application services and distributed computing. Java EE can be application servers which runs at the reference time. It can be deployed with safety, scalability, concurrency.

Java EE is defined by its applications and specific projects. The applications and specific projects defines the application programming interface and interactions. There are many Java Community Process applications and specific projects, so the J2EE have to fulfill some conformance requirements to make the products of other providers conform to Java EE.

In this paper, J2EE projects are chosen because they usually have exactly the same interfaces and methods in order to meet the certain requirement of the other applications which are using J2EE. And the same interfaces and the same methods can provide enough "similar" code pairs which are suitable to be used as training and testing dataset.

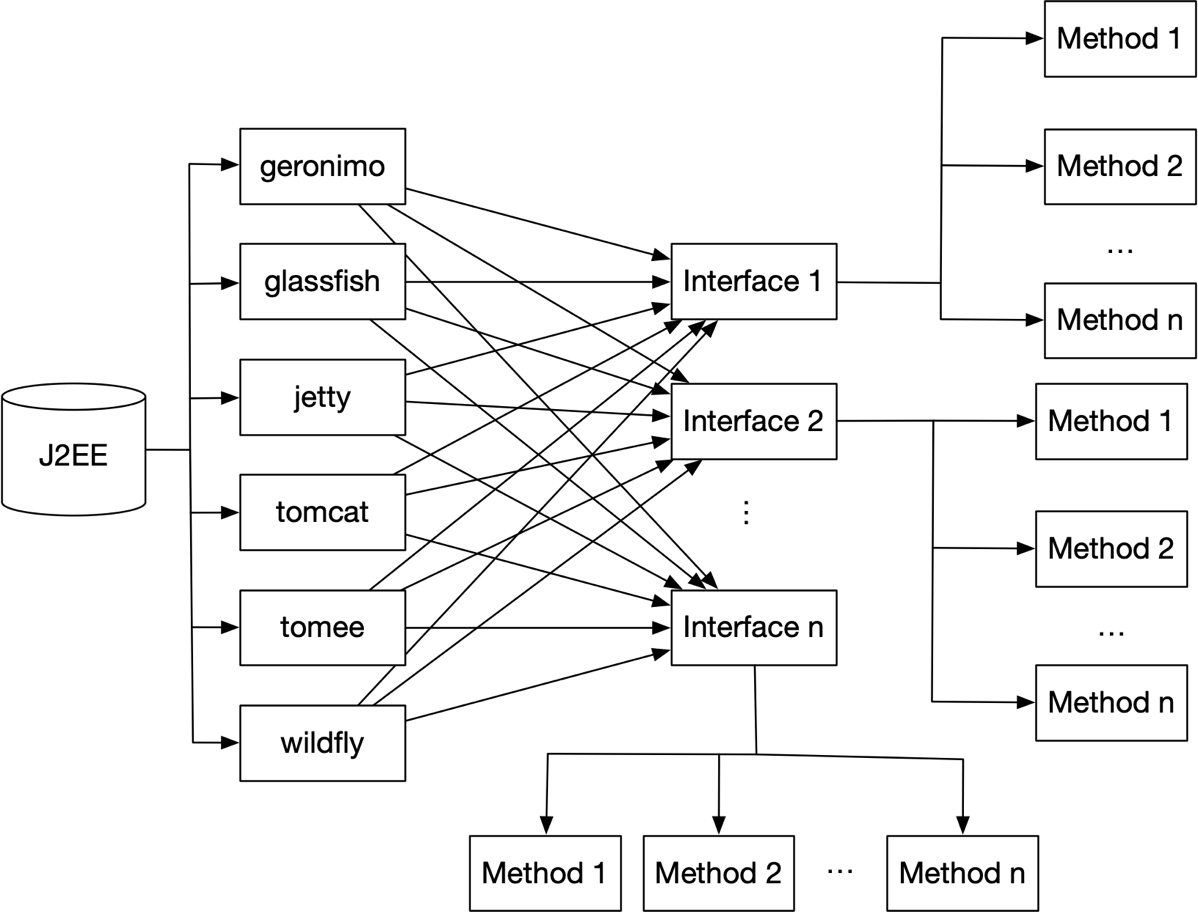


Figure 4.1 Extract similar methods from J2EE

As figure 4.1 shows, in this paper, there are mainly six types of J2EE projects are chosen to generate and create the J2EE dataset. They are Geronimo, Glassfish, Jetty, Tomcat, TomEE and Wildfly.

The first step of making J2EE similar code pairs dataset is to extract the methods which have the similar semantic function. In order to get the similar method, the same interfaces are extracted from the open source J2EE programs. All of the interfaces of J2EE specifications are aiming to realize and achieve the same function, so the methods in the same interfaces usually has the same function.

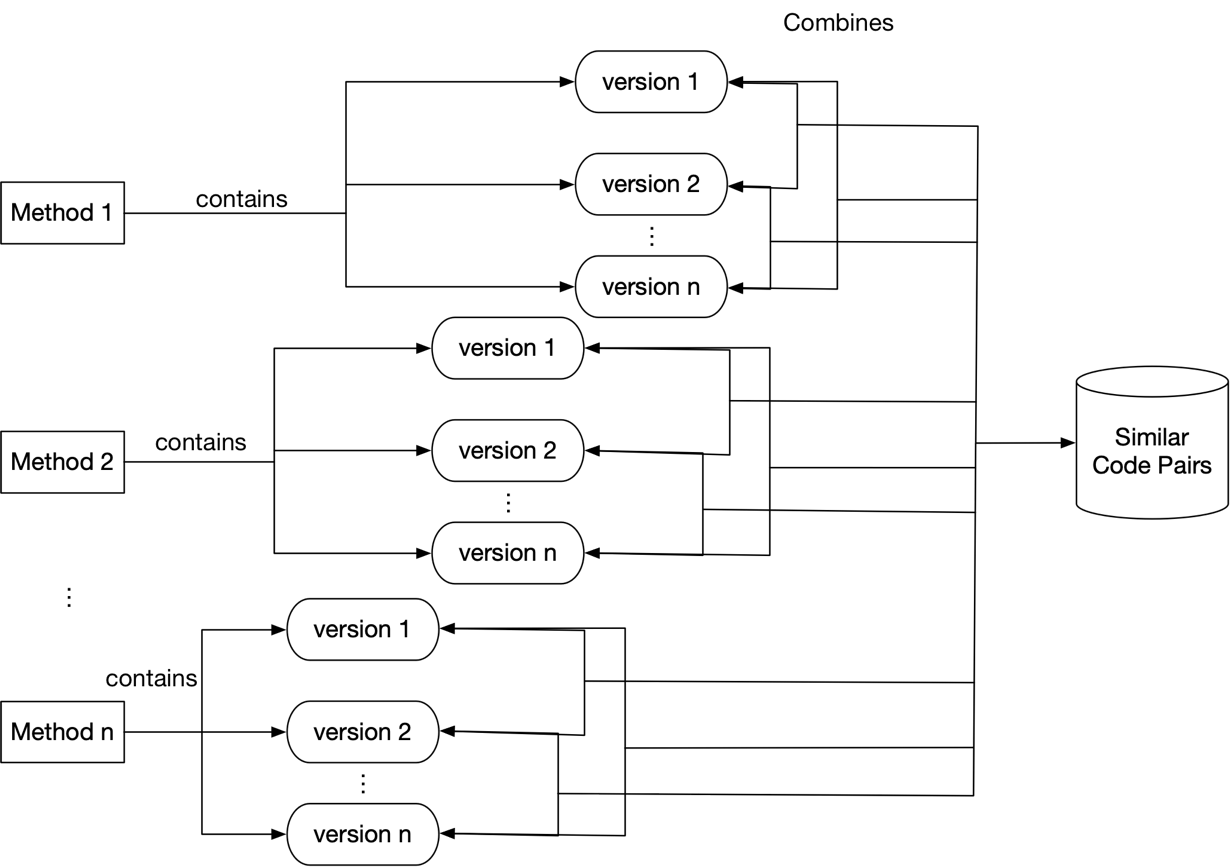


Figure 4.2 Merge every two versions of methods as a dataset

After collecting the similar methods group, the different versions of these methods should be matched one by one which means they should be collected and merged as many pairs. In this process, there are some restrictions that will constrain the combination of method pairs: (1) The length of any method in the method pair cannot exceed 3 times or less than 1/3 of the other method; (2) In the method pairs, the length of any method cannot be less than 2.

Method pairs that do not meet any of the above conditions will be removed. The reason for doing this is to (1) remove pairs of methods whose method lengths are too large, so as to filter out pairs of methods that are not possible because of different method lengths; (2) remove method pairs whose method length is too short, and thus filter out some methods that are not practical and improve the accuracy of the next step model training.

The data after re-pairing will be exponentially larger than the original data. And finally, the pairs of program codes are collected together as J2EE dataset, as figure 4.2 shows. An example of output data of J2EE is shown in figure 4.3.



Figure 4.3 An example of J2EE similar program data

4.1.2 BigCloneBench

BigCloneBench is a clone detection benchmark for known clones in the IJ dataset source repository. BigCloneBench provides the code clone snippets with many levels from one to four. In this paper, only level four code clones in BigCloneBench are used as training and testing data of the CNN model because level four codes can represent the semantic similar of the programs and the codes, whereas level one to three only represent the Similar appearance of the snippets of code.

There is no need to manually extract similar methods from the programs in BigCloneBench because it already provides the code clusters with all clone of levels. Therefore, after getting the methods clusters, every two versions of methods are collected together and merged as a dataset, as figure 4.2 shows. Then the BigCloneBench dataset is finished. Figure 4.4 represents an example of similar programs in BigCloneBench data.

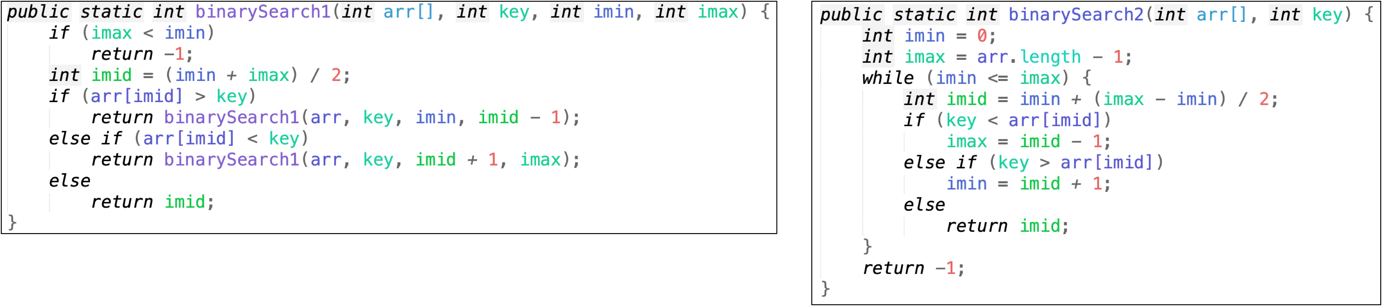


Figure 4.4 An example of similar programs in BigCloneBench

## 4.2 Data Pre-process

The code pairs cannot be used as input directly. In order to make the original code segment pairs smoothly input into the CNN model, pre-processing is necessary.

To begin with, the raw code snippets should be abstracted as code vector, using the AST and Path-Attention model in section 3.2. Additionally, the code vector pairs should be converted and merged as one code vector with two channels, using concatenate function in package NumPy.

## 4.3 Convolutional Neural Network Model

In this section, the detail of the convolutional neural network will be introduced including model framework, model structure and model parameters.

As figure 4.7 shows, the CNN model has eight convolution layers, three max pooling layers two dense layers one flatten layers and one dropout layer.

(1) *Convolution layer*: The term convolution begins with signals and linear systems, and the discussion of signals and linear systems is what happens after a pass through a linear system. Since in reality, the output of a signal at a previous moment often affects the output at that moment, the unit response of the system is generally convolved with the input of the system to obtain the output signal of the system which is required to be linear Time constant.

Convolution is the result of summing two variables over a range. If the convolved variable is the sequence and , then the result of the convolution can be represented as:

(2) *Pooling layer*: The pooling operation makes the model more concerned with the existence of certain features rather than the specific location of the feature. It can be regarded as a strong prior, which makes the feature learning contain a certain degree of freedom and can tolerate some small displacements of features. Due to the down sampling effect of the pooling operation, one element in the pooling result corresponds to a sub-region of the original input data, so the pooling is equivalent to making a dimension reduction in the spatial range, so that the model can extract a wider range of features. At the same time, the input size of the next layer is reduced, thereby reducing the amount of calculation and the number of parameters. To some extent, it prevents the occurrence of overfitting. Figure 4.5 is an example for pooling operation.

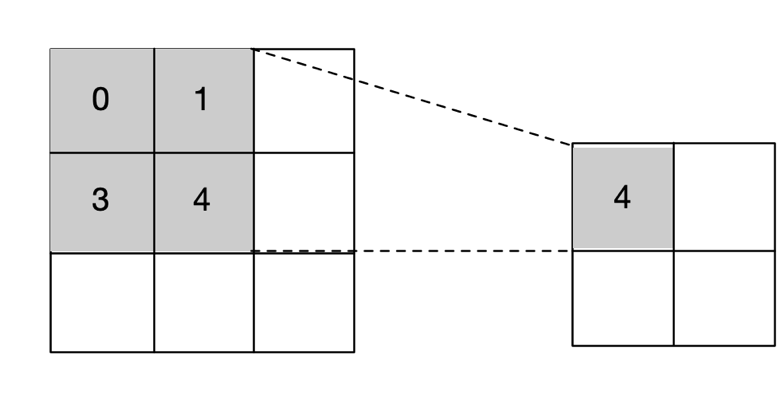


Figure 4.5 Max pooling operation

(3) *Dense layer*: The fully connected layer (Dense layer) acts as a "classifier" throughout the convolutional neural network. If the operation of the convolutional layer, the pooling layer, and the activation function layer is to map the original data to the hidden layer feature space, the fully connected layer plays the role of mapping the learned "distributed feature representation" to the sample mark space.

(4) *Flatten layer*: The Flatten layer is used to "flatten" the input, that is, to multi-dimensionalize the multidimensional input. It is often in the transition from the convolutional layer to the fully connected layer (Dense). That is to say, after the convolution layer is not directly connected to the Dense full connection layer, the data of the Convolution layer (Flatten) is needed to flatten, and then the Dense layer can be directly added.

(5) *Dropout layer*: Dropout can be used as a trick to train deep neural networks. In each training batch, over-fitting can be significantly reduced by ignoring half of the feature detectors (making half of the hidden layer nodes have a value of 0). This approach reduces the interaction between feature detectors (hidden layer nodes), which means that some detectors rely on other detectors to function, as figure 4.6 shows.

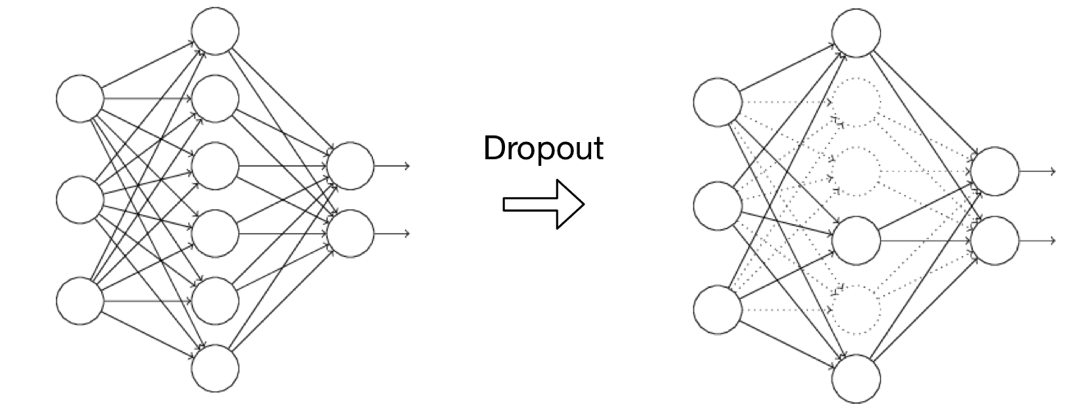


Figure 4.6 Dropout operation

As figure 4.7 shows, in the first convolution layer, the shape of input is (, 2, 384, 1) where 2 is the number of channels, 384 is the length of the vector and 1 means it is a one-dimension vector.

The final output of the dense layer has a shape as (, 2) which means there are only two results: 1 represents of the similar of the pair of code, and 0 represents of the dissimilar of the pair of code.

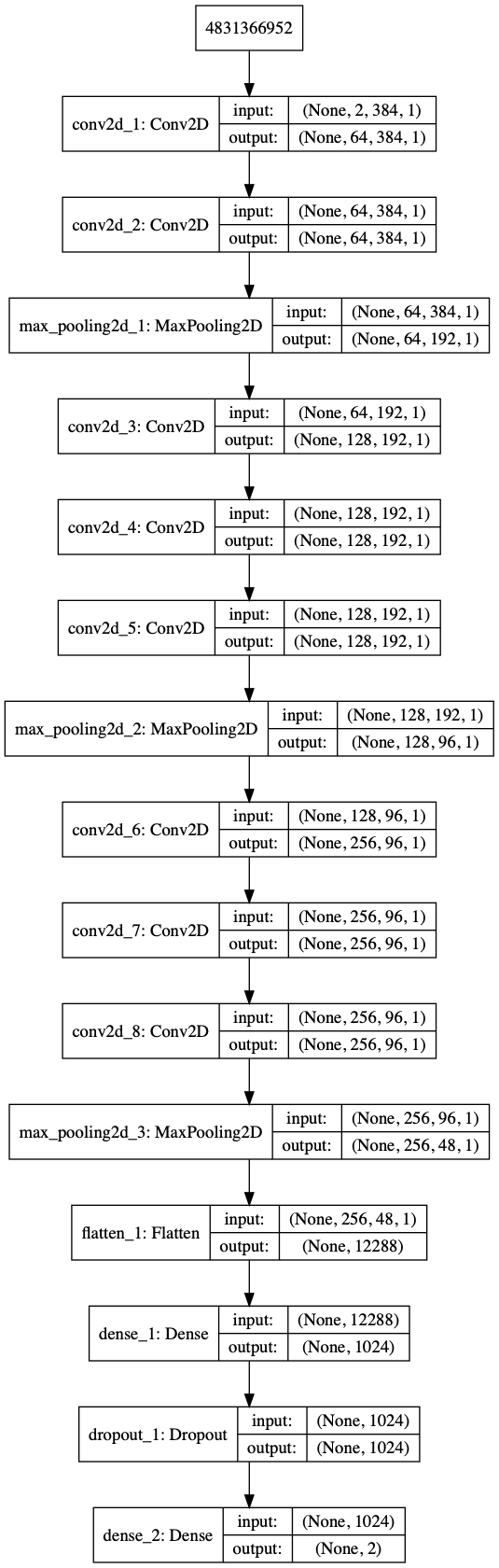


Figure 4.7 Architecture of the CNN model