

ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC BÁCH KHOA

BÁO CÁO MÔN HỌC
KHOA HỌC DỮ LIỆU VÀ TRÍ TUỆ NHÂN TẠO NÂNG CAO
CHỦ ĐỀ: ỨNG DỤNG MẠNG NƠ RON TÍCH
CHẬP TRONG ĐỊNH VỊ LỖI PHẦN MỀM

Nghiên cứu sinh:

Nguyễn Thanh Bình

Cao Thị Nhâm

Đà Nẵng, ngày 26 tháng 04 năm 2023

MỤC LỤC

MỤC LỤC	1
CHƯƠNG 1. ĐẶT VẤN ĐỀ	4
1.1 Tổng quan về đề tài.....	4
1.2 Mô tả bài toán	4
CHƯƠNG 2. CƠ SỞ LÝ THUYẾT	6
2.1 Mạng nơ ron tích chập	6
2.2 Các kỹ thuật xử lý mất cân bằng dữ liệu	9
2.3 Phương pháp định vị lỗi dựa trên phổ	10
2.3.1 Tổng quan	10
2.3.2 Các phương pháp đo lường tính hiệu quả của các phương pháp định vị	
lỗi	11
CHƯƠNG 3. MÔ TẢ DỮ LIỆU	13
CHƯƠNG 4. PHƯƠNG PHÁP THỰC HIỆN	15
4.1 Tổng quan về phương pháp thực hiện	15
4.2 Tiền xử lý dữ liệu.....	16
4.3 Xây dựng kiến trúc mạng nơ ron tích chập	17
4.4 Huấn luyện mô hình.....	18
4.5 Định vị lỗi	19
CHƯƠNG 5. KẾT QUẢ.....	22
5.1 Đánh giá mô hình.....	22
5.2 Đánh giá hiệu quả định vị lỗi.....	23

Danh mục hình ảnh

Hình 2-1 Minh họa kiến trúc cơ bản của CNN	6
Hình 2-2 Minh họa hoạt động của tầng tích chập	7
Hình 2-3 Max pooling	8
Hình 2-4 Average pooling	8
Hình 2-5 Minh họa tầng kết nối đầy đủ.....	9
Hình 4-1 Ma trận độ phủ mã nguồn	16
Hình 4-2 Ví dụ về ma trận độ phủ mã nguồn	16
Hình 4-3 Kiến trúc CNN một chiều	17
Hình 4-4 Mã nguồn thiết lập kiến trúc mạng CNN.....	18
Hình 4-5 Mã nguồn huấn luyện mô hình CNN	19
Hình 4-6 Minh họa cho test case ảo	19
Hình 4-7 Minh họa danh sách giá trị nghi ngờ lỗi và xếp hạng	20

Danh mục bảng biểu

Bảng 3-1 Thống kê lỗi theo các dự án.....	13
Bảng 3-2 Thông tin dự án Chart và Cloure	14
Bảng 4-1 Bảng minh họa danh sách xếp hạng nghi ngờ lỗi của module 8 – dự án Chart	20
Bảng 5-1 Tổng kết kết quả định vị lỗi theo dự án.....	23
Bảng 5-2 Tổng kết kết quả định vị lỗi theo module.....	23

CHƯƠNG 1. ĐẶT VẤN ĐỀ

1.1 Tổng quan về đề tài

Tìm và sửa lỗi là công việc không thể thiếu trong phát triển phần mềm. Định vị lỗi phần mềm đề cập đến việc xác định nguyên nhân và vị trí đoạn mã gây ra lỗi. Lỗi được định vị sớm giúp rút ngắn thời gian sửa lỗi, từ đó giảm thiểu thời gian chết của hệ thống, vì vậy tăng tính sẵn sàng cho các hệ thống. Các kỹ thuật định vị lỗi phần mềm truyền thống dựa trên việc phân tích mã nguồn một cách thủ công nên rất tốn thời gian và chi phí. Quy mô và độ phức tạp của phần mềm ngày càng tăng dẫn tới việc xác định lỗi ngày càng trở nên khó khăn, do vậy cần phải có những kỹ thuật tự động hoặc bán tự động để giảm thiểu công sức con người và rút ngắn thời gian định vị lỗi.

Nhiều phương pháp định vị lỗi đã được các nhà nghiên cứu công bố như phương pháp định vị lỗi dựa trên phổ, phương pháp cắt gọt, phương pháp dựa trên phân tích thông tin,... Tuy nhiên, chưa có phương pháp nào thể hiện ưu thế vượt trội về mặt hiệu năng và độ chính xác của định vị lỗi.

Các kỹ thuật học máy đã được ứng dụng vào nâng cao hiệu suất và độ chính xác của định vị lỗi. Nội dung báo cáo này tập trung thực hiện định vị lỗi dựa trên mạng nơ ron tích chập. Dữ liệu sử dụng trong báo cáo này là Defects4j phiên bản 2.0.

1.2 Mô tả bài toán

Dưới đây là một số khái niệm để làm rõ phạm vi của bài toán.

- **Sự cố (Failures).** Failure is the term used to indicate a system's inability to work according to expectations [43] including unexpected output or incorrect data. Một sự cố xảy ra khi người dùng nhận thấy rằng chương trình ngừng cung cấp dịch vụ như mong đợi.
 - Lỗi khi có kết quả sai lệch so với yêu cầu đặc tả, là sự khác biệt giữa kết quả thực tế trên màn hình và kết quả mong đợi của một thành phần, hệ thống hoặc service nào đó.
- **Lỗi (Errors).** Sự khác biệt giữa giá trị hoặc điều kiện được tính toán, quan sát hoặc đo lường và giá trị hoặc điều kiện thực, được chỉ định hoặc đúng về mặt lý thuyết. Lỗi xảy ra khi một phần nào đó của phần mềm máy tính tạo ra trạng thái không mong muốn.
 - Là hành động của con người dẫn đến kết quả sai.

- **Lỗi (Faults).** Lỗi là một bước hoặc một thủ tục hoặc một định nghĩa dữ liệu không chính xác trong chương trình máy tính do lập trình viên tạo ra
 - Lỗi xảy ra khi làm sai các step, process, hoặc chuẩn bị dữ liệu.

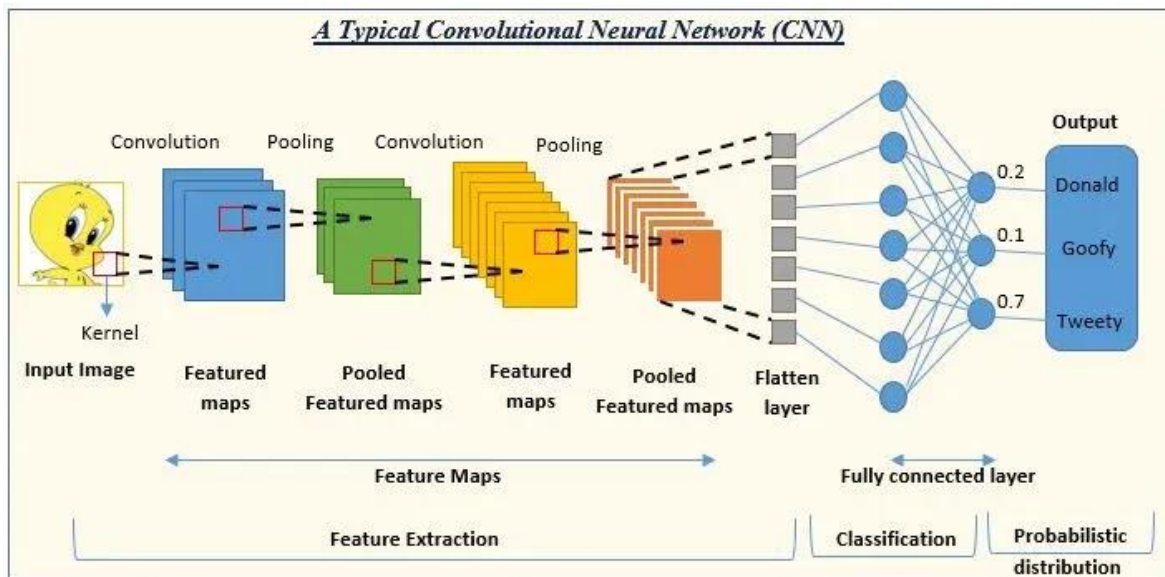
Bài toán định vị lỗi phần mềm được phát biểu như sau: Cho mã nguồn của một dự án phần mềm có chứa các lỗi và các test case, sử dụng thuật toán học sâu để định vị các lỗi trong mã nguồn. Cụ thể hơn, thuật toán cần trả về danh sách các dòng code nghi ngờ bị lỗi theo thứ tự từ cao đến thấp.

CHƯƠNG 2. CƠ SỞ LÝ THUYẾT

2.1 Mạng nơ ron tích chập

CNN, viết tắt của Convolutional Neural Network, là một mô hình Deep Learning tiên tiến. Được biết đến với khả năng xây dựng các hệ thống có độ chính xác cao và thông minh, CNN có nhiều ứng dụng đặc biệt trong việc nhận dạng vật thể trong hình ảnh và các bài toán tương tự.

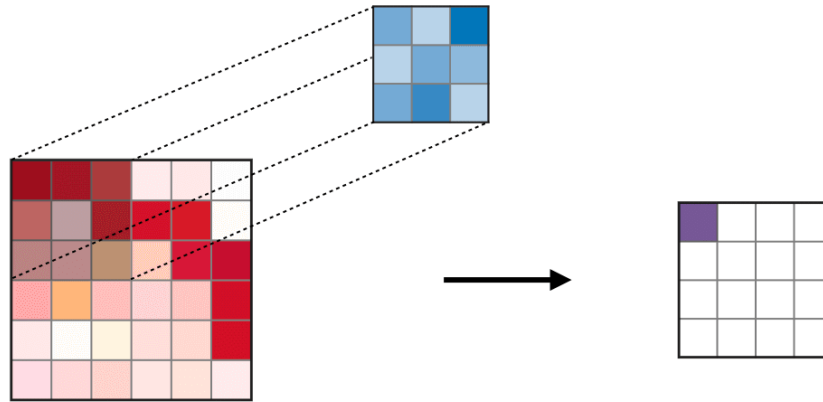
Các lớp cơ bản của CNN được minh họa như Hình 2-1.



Hình 2-1 Minh họa kiến trúc cơ bản của CNN

Tầng tích chập (Convolutional layer)

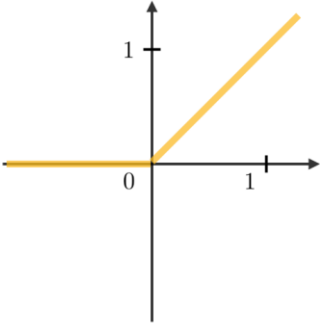
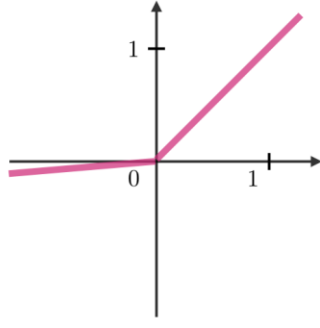
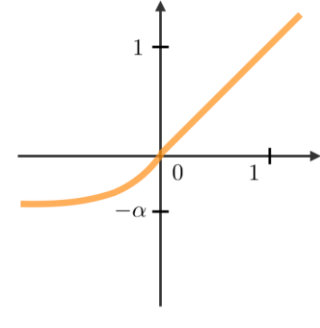
Convolutional là một loại cửa sổ dạng trượt nằm trên một ma trận. Các convolutional layer sẽ chứa các parameter có khả năng tự học, qua đó sẽ điều chỉnh và tìm ra cách lấy những thông tin chính xác nhất trong khi không cần chọn feature. Tầng tích chập (CONV) sử dụng các bộ lọc để thực hiện phép tích chập khi đưa chúng đi qua đầu vào I theo các chiều của nó. Các siêu tham số của các bộ lọc này bao gồm kích thước bộ lọc F và độ trượt (stride) S . Kết quả đầu ra O được gọi là feature map hay activation map.



Hình 2-2 Minh họa hoạt động của tầng tích chập

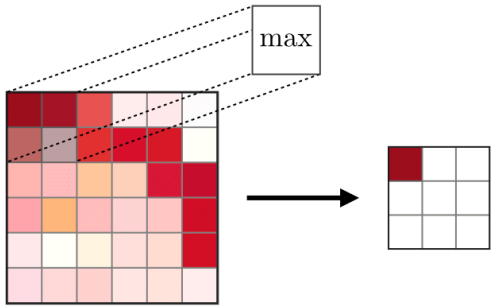
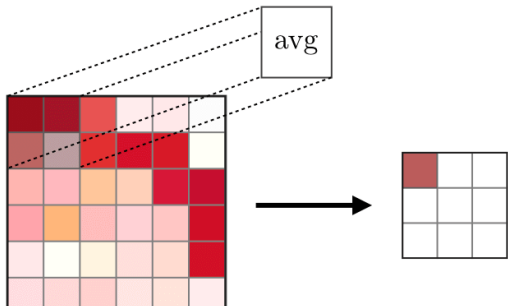
Relu layer

Còn có tên gọi khác là activation function, đây là một hàm được kích hoạt trong neural network. Nó có tác dụng mô phỏng các neuron có tỷ lệ truyền xung qua axon. Trong activation function chúng còn có hàm nghĩa là: Relu, Leaky,... Relu layer được ứng dụng phổ biến trong việc huấn luyện nơ-ron do sở hữu nhiều ưu điểm tiên tiến.

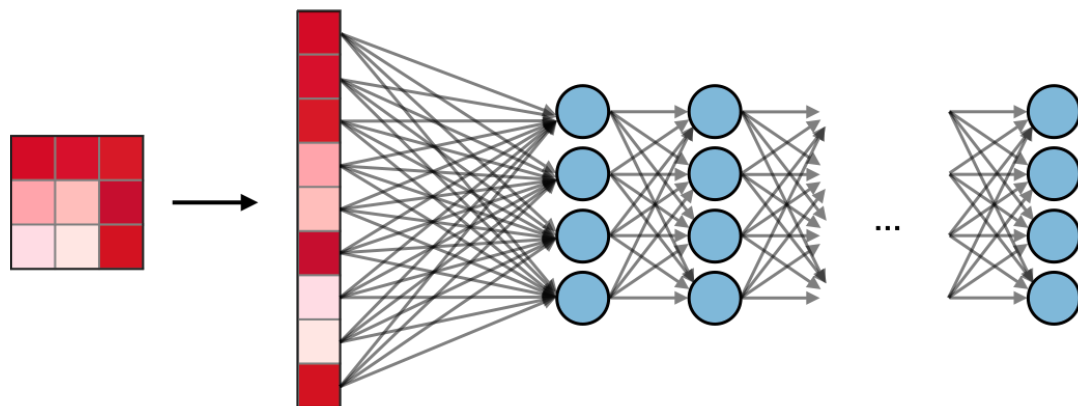
ReLU	LeakyReLU	ELU
$g(z)=\max(0,z)$	$g(z)=\max(\epsilon z, z)$ với $\epsilon \ll 1$ $\epsilon \ll 1$	$g(z)=\max(\alpha(e^z-1), z)$ với $\alpha \ll 1$ $\alpha \ll 1$
		
<ul style="list-style-type: none"> • Độ phức tạp phi tuyến tính có thể thông dịch được về mặt sinh học 	<ul style="list-style-type: none"> • Gán vấn đề ReLU chết cho những giá trị âm 	<ul style="list-style-type: none"> • Khả vi tại mọi nơi

Pooling (POOL).

Tầng pooling (POOL) là một phép downsampling, thường được sử dụng sau tầng tích chập, giúp tăng tính bất biến không gian. Cụ thể, max pooling và average pooling là những dạng pooling đặc biệt, mà tương ứng là trong đó giá trị lớn nhất và giá trị trung bình được lấy ra. Có 2 kiểu pooling: max pooling và average pooling

Max pooling	Average pooling
Từng phép pooling chọn giá trị lớn nhất trong khu vực mà nó đang được áp dụng.	Từng phép pooling tính trung bình các giá trị trong khu vực mà nó đang được áp dụng.
 <p>Hình 2-3 Max pooling</p>	 <p>Hình 2-4 Average pooling</p>
<ul style="list-style-type: none"> • Bảo toàn các đặc trưng đã phát hiện • Được sử dụng thường xuyên 	<ul style="list-style-type: none"> • Giảm kích thước feature map • Được sử dụng trong mạng LeNet

Fully Connected (FC). Tầng kết nối đầy đủ (FC) nhận đầu vào là các dữ liệu đã được làm phẳng, mà mỗi đầu vào đó được kết nối đến tất cả neuron. Trong mô hình mạng CNNs, các tầng kết nối đầy đủ thường được tìm thấy ở cuối mạng và được dùng để tối ưu hóa mục tiêu của mạng ví dụ như độ chính xác của lớp.



Hình 2-5 Minh họa tầng kết nối đầy đủ

2.2 Các kỹ thuật xử lý mất cân bằng dữ liệu

Sự mất cân bằng dữ liệu thường xảy ra trong các bài toán phân loại nhị phân như phát hiện email rác, phát hiện gian lận, dự đoán vỡ nợ, chuẩn đoán bệnh tật, và nhiều ứng dụng khác. Trong trường hợp tỷ lệ dữ liệu giữa hai nhóm là 50:50 thì coi đó là cân bằng. Tuy nhiên, khi có sự chênh lệch giữa hai nhóm, như 60:40, thì ta nói dữ liệu gặp hiện tượng mất cân bằng.

Hầu hết các tập dữ liệu thường khó có thể đạt được sự cân bằng hoàn hảo và luôn có sự chênh lệch về tỷ lệ giữa hai nhóm. Đối với các trường hợp mất cân bằng nhẹ nhàng, như tỷ lệ 60:40, điều này thường không ảnh hưởng đáng kể đến khả năng dự báo của mô hình.

Tuy nhiên, khi mất cân bằng dữ liệu trở nên nghiêm trọng, như tỷ lệ 90:10, thì thường dẫn đến sự hiểu lầm về chất lượng của mô hình. Trong trường hợp này, thước đo chính xác (accuracy) có thể rất cao mà không cần phải sử dụng mô hình. Ví dụ, một dự đoán ngẫu nhiên chỉ chọn nhóm đa số có thể đạt được độ chính xác là 90%. Do đó, không nên sử dụng chính xác làm thước đo đánh giá mô hình để tránh sai lầm lạc quan về chất lượng.

Dưới đây là một số kỹ thuật xử lý việc mất cân bằng dữ liệu:

a) Under-sampling

Under-sampling là quá trình giảm số lượng quan sát trong nhóm đa số để làm cho nó cân bằng với số lượng quan sát trong nhóm thiểu số. Một ưu điểm của phương pháp này

là nó giúp cân bằng mẫu một cách nhanh chóng và dễ dàng thực hiện mà không cần đến các thuật toán phức tạp để tạo mẫu.

Tuy nhiên, một nhược điểm của under-sampling là kích thước mẫu sẽ bị giảm đáng kể. Ví dụ, nếu nhóm thiểu số có kích thước là 500, để đạt được sự cân bằng giữa nhóm đa số và nhóm thiểu số, ta cần giảm kích thước mẫu của nhóm đa số từ 10.000 xuống 500. Kích thước tổng của tập huấn luyện sau khi áp dụng under sampling sẽ là 1.000, chiếm gần 1/10 kích thước ban đầu của tập huấn luyện. Tập huấn luyện mới này có kích thước nhỏ, không phản ánh đúng phân phối của toàn bộ dữ liệu và thường dễ gây ra hiện tượng overfitting.

b) Over-sampling

Over-sampling là các phương pháp giúp giải quyết hiện tượng mất cân bằng mẫu bằng cách gia tăng kích thước mẫu thuộc nhóm thiểu số bằng các kỹ thuật khác nhau. Có 2 phương pháp chính để thực hiện over sampling đó là:

- Lựa chọn mẫu có tái lập.
- Mô phỏng mẫu mới dựa trên tổng hợp của các mẫu cũ.

2.3 Phương pháp định vị lỗi dựa trên phổ

2.3.1 Tổng quan

Trong định vị lỗi phần mềm, có hai loại phân tích: phân tích tĩnh và phân tích động. Phân tích tĩnh được thực hiện thông qua việc kiểm tra mã nguồn chương trình và lý do đối với tất cả các hành vi có thể xảy ra trong thời gian chạy. Kỹ thuật này đã thành công do việc sử dụng nó không gây khó chịu và không cần chạy chương trình [1]. Phân tích động [2] được thực hiện thông qua việc chạy chương trình. Trong thực tế, nhiều trường hợp kiểm thử (test case) được viết để đảm bảo các yêu cầu của chương trình được đáp ứng. Trường hợp kiểm thử thất bại dẫn đến việc phát hiện ra lỗi trong mã chương trình. Thông tin về phạm vi mã có thể được ghi lại bằng cách sử dụng phân tích động. Phân tích chỉ ra mức độ mã chương trình đã được thực thi bởi các trường hợp kiểm thử. Các loại phạm vi mã khác nhau có thể được sử dụng trong phân tích động để xác định các lỗi tiềm ẩn trong mã chương trình. Những loại này là các câu lệnh, khối, hàm, vị từ và đường dẫn của chương trình. Kỹ thuật phân tích dựa trên phổ (SBFL - Spectrum-Based Fault Localization) được công bố đầu tiên trong bài báo [3]. Tác giả đề xuất rằng phổ chương trình, chẳng hạn như phạm vi mã, thông tin kiểm tra, dấu vết thực thi, đường dẫn thực thi, hồ sơ đường dẫn và hồ sơ thực

thi [4] [5] có thể được sử dụng trong việc tìm kiếm các vị trí lỗi. Điều này có thể được thực hiện bằng cách so sánh các phần tử trong phổ chương trình sau khi thực hiện kiểm tra, để suy ra và xác định vị trí lỗi.

Dựa trên ý tưởng này, SBFL phân tích thông tin kiểm thử (tức là lỗi và thành công của quá trình thực hiện kiểm thử) cho một chương trình dựa trên việc thực hiện các trường hợp kiểm thử để xác định các vị trí đáng ngờ có thể chứa lỗi. Bằng cách sử dụng công thức (ví dụ: Tarantula, Ochiai, Naish, v.v.) hoặc các phương trình liên quan đến quy tắc xác suất, thông tin kiểm thử sẽ được sử dụng để tính toán giá trị xác định khả năng xác định vị trí lỗi. Điều này được gọi là mức độ nghi ngờ lỗi có giá trị từ 0 đến 1, trong đó giá trị 1 thể hiện khả năng cao nhất hoặc đáng ngờ nhất là vị trí lỗi và ngược lại.

Khi một phát biểu có nhiều khả năng mắc lỗi hoặc có yếu tố sai sót hơn phát biểu khác thì phát biểu đó thường có điểm nghi ngờ cao hơn [5]. Các giá trị đáng ngờ được tính toán dựa trên tần suất sử dụng các câu lệnh trong các trường hợp kiểm thử đạt hoặc không đạt. Một phần tử được coi là đáng ngờ nếu nó được chạy thường xuyên hơn trong các thử nghiệm thất bại và ít thường xuyên hơn trong các thử nghiệm thành công.

Nhiều phương pháp tiếp cận để định vị lỗi dựa trên phổ đã được đề xuất trong tài liệu [6] [7] [8] [9] [10].

2.3.2 Các phương pháp đo lường tính hiệu quả của các phương pháp định vị lỗi

a) Top-k

Top-n là số lỗi liên quan được xếp hạng trong K ($K = 1, 5, 10$) hàng đầu trong kết quả trả về. Với một báo cáo lỗi, nếu K kết quả truy vấn hàng đầu chứa ít nhất một tệp cần sửa lỗi, thì coi đó là lỗi đã được định vị. Giá trị số liệu càng cao thì kỹ thuật định vị lỗi sẽ hoạt động càng tốt.

b) Mean Reciprocal Rank (MRR)

MRR là độ đo phổ biến được sử dụng để đánh giá kỹ thuật định vị lỗi, dùng cho kỹ thuật IR (Information Retrieval) là chủ yếu. Cho một truy vấn (giả sử là báo cáo lỗi b), thứ hạng đối ứng của nó là nghịch đảo nhân với thứ hạng của tài liệu đứng đầu tiên (giả sử là tệp mã nguồn) trong danh sách xếp hạng được tạo bởi kỹ thuật xếp hạng (trong trường hợp của chúng tôi là IR- kỹ thuật định vị lỗi dựa trên). MRR là mức trung bình của thứ hạng tương hỗ của tất cả các lỗi trong bộ báo cáo lỗi BR, được tính như sau:

$$\text{MRR}(R) = \frac{1}{|\text{BR}|} \sum_{b \in \text{BR}} \frac{1}{\text{rank}(b)}$$

Trong công thức trên, xếp $\text{rank}(b)$ đề cập đến vị trí của tệp mã nguồn được đề xuất chính xác đầu tiên trong danh sách xếp hạng được kỹ thuật bản địa hóa lỗi dựa trên IR trả về cho báo cáo lỗi b .

c) EXAM

$$\text{EXAM} = \frac{\text{Thứ tự của thành phần mã nguồn trong danh sách nghi ngờ}}{\text{Số lượng thành phần mã nguồn trong chương trình}}$$

Kỹ thuật nào có EXAM càng cao thì hiệu năng càng tốt và ngược lại.

CHƯƠNG 3. MÔ TẢ DỮ LIỆU

Dữ liệu sử dụng trong báo cáo này là Defects4j [11] – một cơ sở dữ liệu về các lỗi chuyên phục vụ cho các nghiên cứu nâng cao về công nghệ phần mềm.

Defects4j bao gồm 835 lỗi sưu tập từ các dự án mã nguồn mở. Mỗi dự án trong Defects4J có các phiên bản khác nhau của mã nguồn, bao gồm cả phiên bản ban đầu và phiên bản sau khi sửa lỗi. Điều này cho phép các nhà nghiên cứu so sánh và phân tích sự thay đổi trong mã nguồn trước và sau khi sửa lỗi, từ đó giúp cải thiện các phương pháp kiểm thử và sửa lỗi phần mềm.

Mỗi lỗi trong Defects4J được mô tả chi tiết, bao gồm mã lỗi, các bản vá tương ứng, và thông tin về nguyên nhân gây ra lỗi. Thông tin này giúp các nhà nghiên cứu hiểu rõ hơn về các vấn đề cụ thể trong mã nguồn và cách sửa lỗi hiệu quả.

Dưới đây là thống kê các lỗi theo từng dự án.

Bảng 3-1 Thống kê lỗi theo các dự án

Mã dự án	Dự án	Số lượng lỗi	Mã lỗi	Mã lỗi không dùng(*)
Chart	jfreechart	26	1-26	None
Cli	commons-cli	39	1-5,7-40	6
Closure	closure-compiler	174	1-62,64-92,94-176	63,93
Codec	commons-codec	18	1-18	None
Collections	commons-collections	4	25-28	1-24
Compress	commons-compress	47	1-47	None
Csv	commons-csv	16	1-16	None
Gson	gson	18	1-18	None
JacksonCore	jackson-core	26	1-26	None

Mã dự án	Dự án	Số lượng lỗi	Mã lỗi	Mã lỗi không dùng(*)
JacksonDatabind	jackson-databind	112	1-112	None
JacksonXml	jackson-dataformat-xml	6	1-6	None
Jsoup	jsoup	93	1-93	None
JXPath	commons-jxpath	22	1-22	None
Lang	commons-lang	64	1,3-65	2
Math	commons-math	106	1-106	None
Mockito	mockito	38	1-38	None
Time	joda-time	26	1-20,22-27	21

Báo cáo này sử dụng các lỗi của dự án Chart và Cloure. Thông tin của dự án được mô tả vắn tắt trong Bảng 3-2.

Bảng 3-2 Thông tin dự án Chart và Cloure

Dự án	Số lượng lỗi	KLOC	Test cases
Chart	26	96	2205
Closure	133	90	7927

Danh sách các lỗi của 2 dự án được liệt kê trong phần Phụ lục.

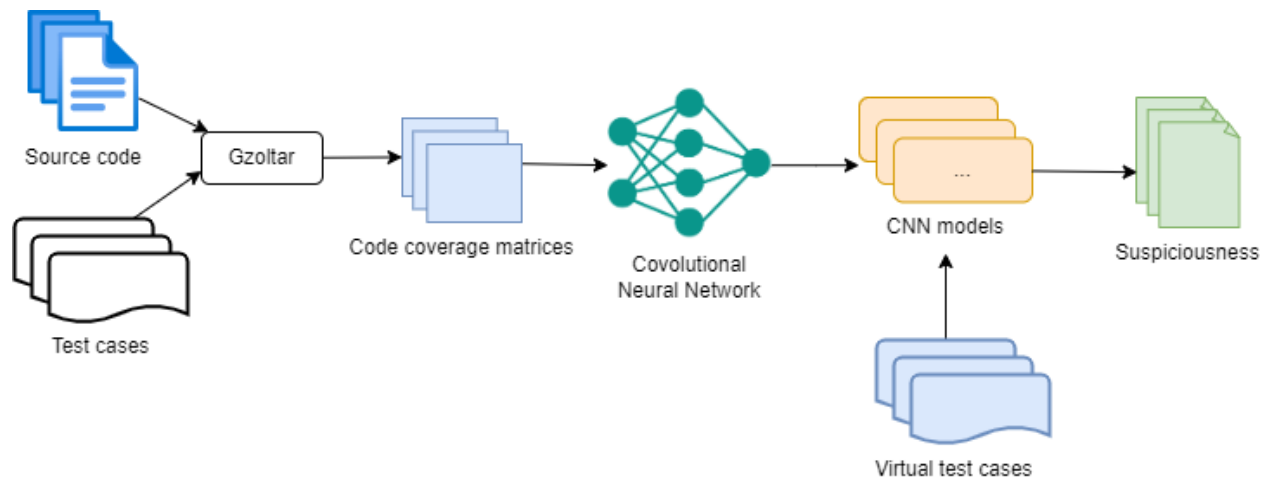
CHƯƠNG 4. PHƯƠNG PHÁP THỰC HIỆN

4.1 Tổng quan về phương pháp thực hiện

CNN được huấn luyện từ các đặc trưng dữ liệu có tính trừu tượng cao từ tập dữ liệu đầu vào và khả năng học của nó sẽ phù hợp để đánh giá những thành phần mã nguồn (dòng lệnh hoặc hàm ...) nào có khả năng gây ra lỗi. Do đó, ý tưởng cơ bản trong bài báo cáo này là áp dụng mạng nơ-ron tích chập để xây dựng mô hình để học và ước tính mối liên hệ của một thành phần mã nguồn với các lỗi. Kết quả đầu ra của mô hình CNN là một giá trị nằm trong khoảng từ 0 đến 1 thể hiện mức độ nghi ngờ lỗi. Giá trị càng lớn thì mức độ nghi ngờ lỗi càng cao.

Quá trình định vị lỗi bắt đầu từ một lỗi và tập trung vào thành phần mã nguồn nào gây ra lỗi.

Quy trình thực hiện được mô tả vắn tắt như Hình 4-1.



Hình 4-1 Quy trình thực hiện định vị lỗi

- **Bước 1.** Xây dựng các ma trận độ phủ mã nguồn dựa trên mã nguồn và các test case.
- **Bước 2.** Xây dựng mạng CNN có kiến trúc mô tả như mục 4.3
- **Bước 3.** Huấn luyện mô hình sử dụng ma trận độ phủ mã nguồn và vector lỗi. Mỗi ma trận được huấn luyện để tạo ra một mô hình. Trong phạm vi bài báo cáo này, chúng tôi sử dụng mini batch có kích thước bằng 10.
- **Bước 4.** Mỗi module, tạo một bộ test case ảo sao cho mỗi test case ảo chạy qua một thành phần mã nguồn.

- **Bước 5.** Sử dụng mô hình đã được huấn luyện để phỏng đoán kết quả đầu ra cho các test case ảo. Giá trị đầu ra chính là mức độ nghi ngờ lỗi của thành phần mã nguồn. Giá trị càng cao thì mức độ nghi ngờ lỗi càng lớn. Sắp xếp mức độ nghi ngờ từ cao tới thấp.

4.2 Tiên xử lý dữ liệu

Công cụ Gzoltar để sinh ra các ma trận độ phủ mã nguồn hay ma trận phổ (spectrum data). Với Một chương trình có N thành phần mã nguồn và M test case, sau khi sử dụng công cụ Gzoltar, kết quả thu được một ma trận độ phủ gồm có N cột và M dòng. Trong đó, giá trị x_{ij} của ma trận thể hiện thành phần mã nguồn i có được thực thi bởi test case j hay không (1 là có được thực thi, 0 là không được thực thi). Vector e_1, e_2, \dots, e_M là kết quả thực hiện test case (1 là thất bại, 0 là thành công) (sau đây sẽ gọi là vector lỗi).

$$\begin{bmatrix} x_{11} & x_{12} & \dots & x_{1N} \\ x_{21} & x_{22} & \dots & x_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ x_{M1} & x_{M2} & \dots & x_{MN} \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_M \end{bmatrix}$$

Hình 4-2 Ma trận độ phủ mã nguồn

Hình 4-2 là ví dụ minh họa cho việc biểu diễn độ phủ mã nguồn.

Program P															Bug information			
S ₁ :Read(a,b,c)			S ₈ : d2 = c+1;			S ₁₅ :else {output(d2);									S3 is faulty. Correct form: If(b<6){			
S ₂ :d1=0,d2=0,d3=0;			S ₉ :if(a < 0){			S ₁₆ :output(d3);}												
S ₃ :if(b < 0){			S ₁₀ :a = a+c;}															
S ₄ :d1 = b;			S ₁₁ : else a = a+b;															
S ₅ :d2 = c;			S ₁₂ : d3 = a+1;}															
S ₆ :d3 = a;}			S ₁₃ :if(c>0){															
S ₇ :else {d1 = b+1;			S ₁₄ :output(d1);}															
test	a,b,c	S ₁	S ₂	S ₃	S ₄	S ₅	S ₆	S ₇	S ₈	S ₉	S ₁₀	S ₁₁	S ₁₂	S ₁₃	S ₁₄	S ₁₅	S ₁₆	result
t1	-1,5,3	1	1	1	0	0	0	1	1	1	1	0	1	1	1	0	0	1
t2	-2,-7,5	1	1	1	1	1	1	0	0	0	0	0	0	1	1	0	0	0
t3	5,-6,-8	1	1	1	1	1	1	0	0	0	0	0	0	1	0	1	1	0
t4	-5,8,-8	1	1	1	0	0	0	1	1	1	1	0	1	1	0	1	1	0
t5	4,7,11	1	1	1	0	0	0	1	1	1	0	1	1	1	1	0	0	0
t6	4,2,1	1	1	1	0	0	0	1	1	1	0	1	1	1	1	0	0	1

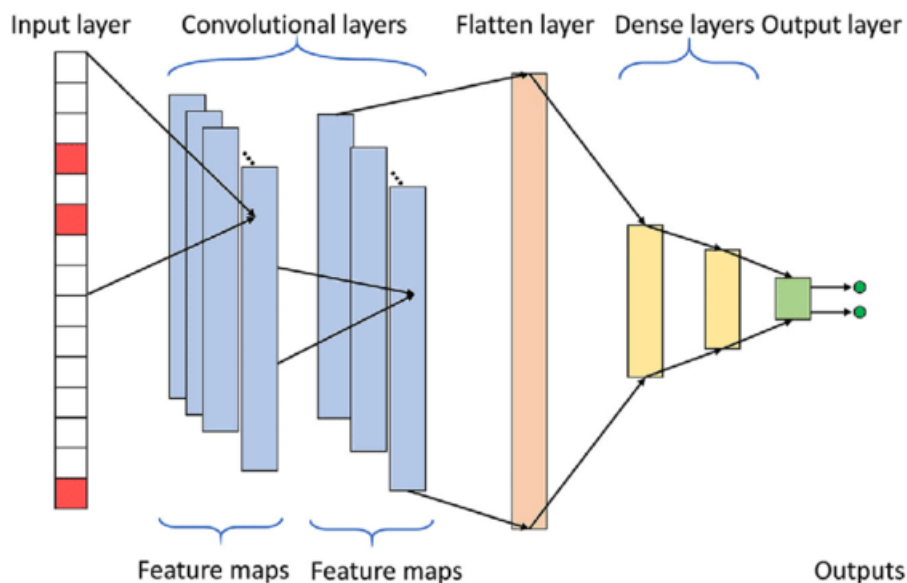
Hình 4-3 Ví dụ về ma trận độ phủ mã nguồn

Trong phạm vi báo cáo này, 2 module đầu tiên của dự án Closure và 8 module đầu tiên của dự án jfreechart.

4.3 Xây dựng kiến trúc mạng nơ ron tích chập

Mỗi module của chương trình sau khi dùng công cụ Gzoltar sẽ sinh ra một ma trận độ phủ mã nguồn. Mô hình CNN một chiều xây dựng cho mỗi ma trận để học các đặc trưng của ma trận đó. Kết quả, một chương trình có N module thì xây dựng được N mô hình CNN.

Hình 4-3 là kiến trúc mô hình CNN một chiều áp dụng cho đề tài này.



Hình 4-4 Kiến trúc CNN một chiều

Lớp input bao gồm ma trận độ phủ mã nguồn kích thước $M \times N$ và vector lỗi. Kích thước của một module lớn, do vậy số lượng phần tử trong mỗi hàng của ma trận cũng rất lớn. Chính vì vậy, cần phải giảm số lượng tham số nhưng vẫn phải giữ lại những đặc trưng quan trọng. Đó là lý do cần tới lớp tích chập.

Thông qua các tích chập giữa ma trận đầu vào với bộ lọc để tạo thành các đơn vị trong một tầng mới. Quá trình này có thể diễn ra liên tục ở phần đầu của mạng và sử dụng kèm với hàm kích hoạt ReLU. Mục tiêu của tầng này là trích xuất đặc trưng một chiều của ma trận độ phủ mã nguồn.

Quá trình tổng hợp được thực hiện ở tầng max pooling: Các tầng càng về sau khi trích xuất đặc trưng sẽ cần số lượng tham số lớn do chiều sâu được quy định bởi số lượng các kênh ở các tầng sau thường tăng tiến theo cấp số nhân. Điều đó làm tăng số lượng tham số và khối lượng tính toán trong mạng nơ ron. Do đó để giảm tải tính toán chúng ta sẽ cần

giảm kích thước các chiều của khối ma trận đầu vào hoặc giảm số đơn vị của tầng. Vì mỗi một đơn vị sẽ là kết quả đại diện của việc áp dụng một bộ lọc để tìm ra một đặc trưng cụ thể nên việc giảm số đơn vị sẽ không khả thi. Giảm kích thước khối ma trận đầu vào thông qua việc tìm ra một giá trị đại diện cho mỗi một vùng không gian mà bộ lọc đi qua sẽ không làm thay đổi các đường nét chính của bức ảnh nhưng lại giảm được kích thước của ảnh. Do đó quá trình giảm chiều ma trận được áp dụng. Quá trình này gọi là tổng hợp nhằm mục đích giảm kích thước.

Quá trình kết nối hoàn toàn (fully connected): Sau khi đã giảm kích thước đến một mức độ hợp lý, ma trận cần được trải phẳng (flatten) thành một vector và sử dụng các kết nối hoàn toàn giữa các tầng. Quá trình này sẽ diễn ra cuối mạng CNN và sử dụng hàm kích hoạt là ReLU. Tầng kết nối hoàn toàn cuối cùng (fully connected layer) sẽ có số lượng đơn vị bằng với số classes và áp dụng hàm kích hoạt là sigmoid nhằm mục đích tính phân phối xác suất.

Dưới đây là mã nguồn của kiến trúc mạng CNN triển khai cho bài báo cáo.

```
def create_model(input_shape):  
    model = tf.keras.Sequential([  
        tf.keras.layers.Input(shape=(input_shape, 1)),  
        tf.keras.layers.Conv1D(32, kernel_size=3, activation='relu'),  
        tf.keras.layers.MaxPooling1D(pool_size=2),  
        tf.keras.layers.Conv1D(64, kernel_size=3, activation='relu'),  
        tf.keras.layers.MaxPooling1D(pool_size=2),  
        tf.keras.layers.Flatten(),  
        tf.keras.layers.Dense(128, activation='relu'),  
        tf.keras.layers.Dense(1, activation='sigmoid') # Output là 1 nếu fail, 0 nếu pass  
    ])  
  
    model.compile(loss='binary_crossentropy',  
                  optimizer=RMSprop(learning_rate=1e-4),  
                  metrics=['accuracy'])  
  
    return model
```

Hình 4-5 Mã nguồn thiết lập kiến trúc mạng CNN

4.4 Huấn luyện mô hình

Dữ liệu đầu vào rất mất cân bằng. Số lượng test case thất bại nhỏ hơn rất nhiều so với số lượng test case thành công. Do đó, trước khi đưa dữ liệu vào huấn luyện, nhóm

ngiên cứu đã thực hiện xử lý mất cân bằng dữ liệu bằng kỹ thuật oversample - SMOTE (Synthetic Minority Over-sampling).

Dữ liệu huấn luyện và đánh giá mô hình chia theo tỉ lệ 80:20 (80% dữ liệu dùng để huấn luyện, 20% để đánh giá).

Các mô hình được lưu dưới dạng file h5.

```
sm = smote.RandomOverSampler(random_state=42)
X, y = sm.fit_resample(spectrum_data, spectrum_labels)

tf.keras.backend.clear_session()

model = create_model(n_elements)
history = model.fit(X, y, epochs=EPOCHS, batch_size=1, shuffle=True, validation_split=0.2)
model.save(save_format='h5', filepath=os.path.join(model_dir, '%s-%s-cnn.h5'%(project, bug)))
```

Hình 4-6 Mã nguồn huấn luyện mô hình CNN

4.5 Định vị lỗi

Với mỗi module, xây dựng tập hợp các test case ảo sao cho mỗi test case chạy qua một thành phần của mã nguồn.

$$\begin{matrix} & x_1 & x_2 & \cdots & x_N \\ \begin{matrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{matrix} & \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \end{matrix}$$

Hình 4-7 Minh họa cho test case ảo

Dùng mô hình đã được huấn luyện để phỏng đoán kết quả đầu ra cho các test case ảo. Giá trị đầu ra là khả năng sinh ra lỗi của thành phần mã nguồn. Giá trị đầu ra càng lớn thì khả năng lỗi càng cao. Danh sách thành phần mã nguồn nghi ngờ lỗi được lập bằng cách sắp xếp chúng theo mức độ nghi ngờ lỗi từ cao tới thấp.

Hình 4-7 là ví dụ minh họa cho kết quả sau khi chạy các test case ảo, kết quả thu được là danh sách các giá trị nghi ngờ lỗi. Dòng lệnh s3 có giá trị nghi ngờ lớn nhất, do vậy sẽ đứng đầu danh sách các lệnh nghi ngờ lỗi.

Program P																Bug information		
S ₁ :Read(a,b,c) S ₂ :d1=0,d2=0,d3=0; S ₃ :if(b < 0){ S ₄ :d1 = b; S ₅ :d2 = c; S ₆ :d3 = a;} S ₇ :else {d1 = b+1; S ₈ : d2 = c+1; S ₉ :if(a < 0){ S ₁₀ :a = a+c;} S ₁₁ : else a = a+b; S ₁₂ : d3 = a+1;} S ₁₃ :if(c>0){ S ₁₄ :output(d1);}																S ₃ is faulty. Correct form: If(b<6){		
test	a,b,c	S ₁	S ₂	S ₃	S ₄	S ₅	S ₆	S ₇	S ₈	S ₉	S ₁₀	S ₁₁	S ₁₂	S ₁₃	S ₁₄	S ₁₅	S ₁₆	result
t1	-1,5,3	1	1	1	0	0	0	1	1	1	1	0	1	1	1	0	0	1
t2	-2,-7,5	1	1	1	1	1	1	0	0	0	0	0	0	1	1	0	0	0
t3	5,-6,-8	1	1	1	1	1	1	0	0	0	0	0	0	1	0	1	1	0
t4	-5,8,-8	1	1	1	0	0	0	1	1	1	1	0	1	1	0	1	1	0
t5	4,7,11	1	1	1	0	0	0	1	1	1	0	1	1	1	1	0	0	0
t6	4,2,1	1	1	1	0	0	0	1	1	1	0	1	1	1	1	0	0	1
suspiciousness		0.43	0.31	0.59	0.34	0.41	0.33	0.48	0.51	0.56	0.44	0.46	0.36	0.35	0.45	0.42	0.49	
Rank		9	16	1	14	11	15	5	3	2	8	6	12	13	7	10	4	

Hình 4-8 Minh họa danh sách giá trị nghi ngờ lỗi và xếp hạng

Bảng 4-1 là minh họa về danh sách thành phần mã nguồn nghi ngờ lỗi của module 9. Dòng lệnh org.jfree.data.general.SeriesException#61 có mức độ nghi ngờ lỗi cao nhất.

Bảng 4-1 Bảng minh họa danh sách xếp hạng nghi ngờ lỗi của module 8 – dự án Chart

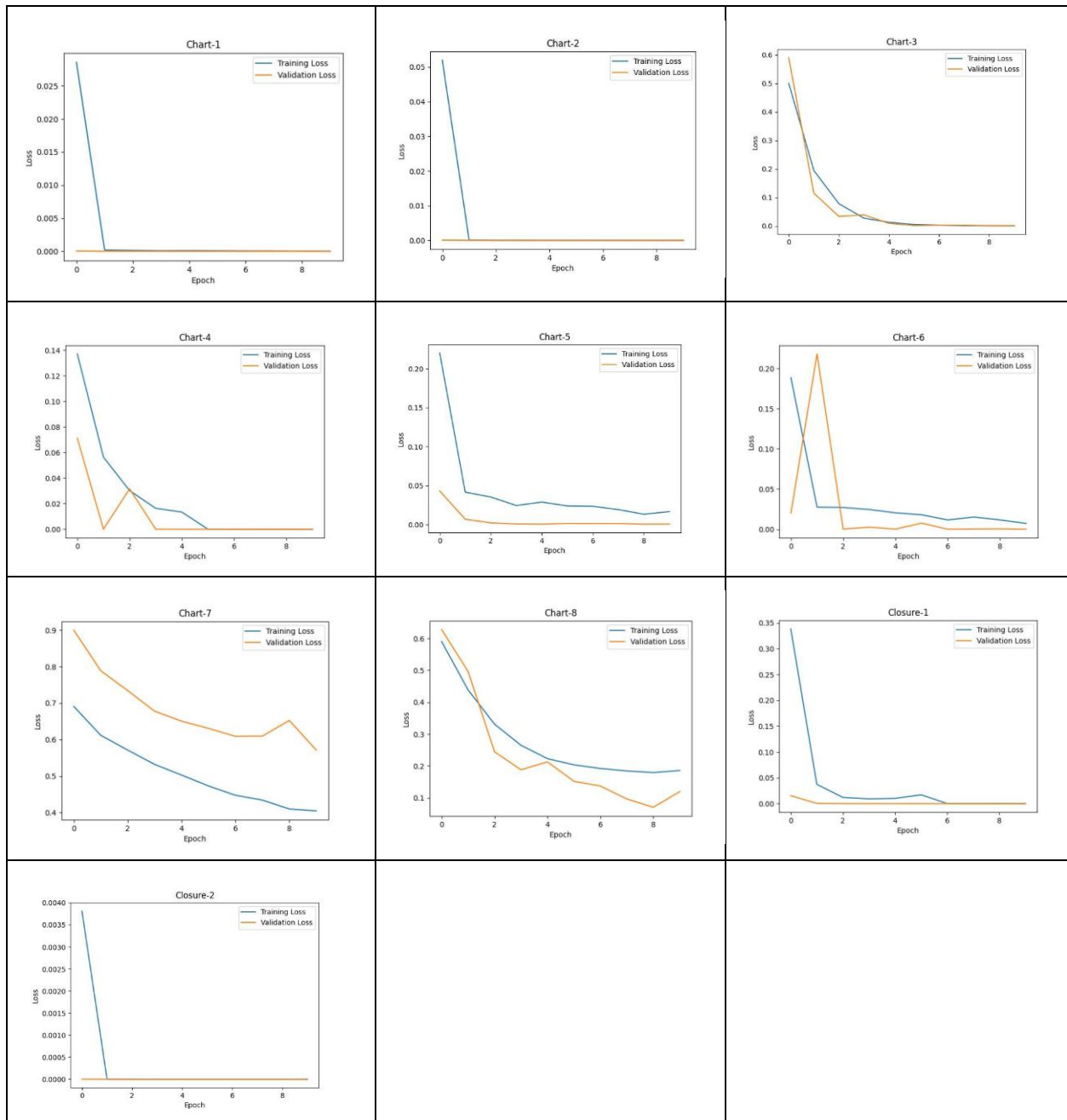
Statement	Suspiciousness
org.jfree.data.general.SeriesException#61	[0.3265599]
org.jfree.data.general.SeriesException#62	[0.3189737]
org.jfree.data.general.SeriesChangeEvent#61	[0.31071335]
org.jfree.data.general.SeriesChangeEvent#62	[0.3059337]
org.jfree.data.general.Series#102	[0.28597832]
org.jfree.data.general.Series#103	[0.27786225]
org.jfree.data.general.Series#111	[0.2712551]
org.jfree.data.general.Series#112	[0.27154234]
org.jfree.data.general.Series#113	[0.26958224]
org.jfree.data.general.Series#115	[0.26598838]
org.jfree.data.general.Series#116	[0.27167377]
org.jfree.data.general.Series#117	[0.27228007]
org.jfree.data.general.Series#118	[0.2710465]
org.jfree.data.general.Series#119	[0.26992422]
org.jfree.data.general.Series#120	[0.27424264]

org.jfree.data.general.Series#130	[0.2754664]
org.jfree.data.general.Series#142	[0.27785927]
org.jfree.data.general.Series#143	[0.27958515]
org.jfree.data.general.Series#145	[0.28788966]
org.jfree.data.general.Series#146	[0.28749794]
org.jfree.data.general.Series#147	[0.28392628]

CHƯƠNG 5. KẾT QUẢ

5.1 Kết quả mô hình

Báo cáo đã xây dựng 10 mô hình tương ứng với 10 module. Độ chính xác của các mô hình từ 99.85% đến 100%. Dưới đây là giá trị hàm mất mát của các mô hình qua 10 epoch.



5.2 Đánh giá hiệu quả định vị lỗi

Để đánh giá hiệu quả định vị lỗi khi áp dụng mạng nơ ron tích chập, hai kỹ thuật truyền thống được mang ra so sánh, đó là Tarantula và Ochiai và Dstar2.

Bảng 5-1 tổng kết số lượng lỗi xác định chính xác theo từng dự án. Kết quả cho thấy, phương pháp CNN mang lại kết quả tốt hơn so với các phương pháp còn lại.

Bảng 5-1 Tổng kết kết quả định vị lỗi theo dự án

Dự án	Kỹ thuật	Top 5	Top 10	Top 15
Closure	Ochiai	0	0	1
	Tarantula	0	0	1
	Dstar2	0	0	1
	CNN	0	0	0
Chart	Ochiai	0	2	2
	Tarantula	0	2	2
	Dstar2	0	2	2
	CNN	0	2	3

Bảng 5-2 thống kê chi tiết số lỗi định vị chính xác theo từng module.

Bảng 5-2 Tổng kết kết quả định vị lỗi theo module

Module	Kỹ thuật	Top 5	Top 10	Top 15
Cloure-1	Ochiai	0	0	0
	Tarantula	0	0	0
	Dstar2	0	0	0
	CNN	0	0	0
Closure-2	Ochiai	0	0	1
	Tarantula	0	0	1
	Dstar2	0	0	1
	CNN	0	0	0
Chart-1	Ochiai	0	0	0
	Tarantula	0	0	0
	Dstar2	0	0	0
	CNN	0	1	1
Chart-2	Ochiai	0	0	0
	Tarantula	0	0	0
	Dstar2	0	0	0
	CNN	0	0	0

Chart-3	Ochiai	0	1	1
	Tarantula	0	1	1
	Dstar2	0	1	1
	CNN	0	1	1
Chart-4	Ochiai	0	1	0
	Tarantula	0	0	0
	Dstar2	0	0	0
	CNN	0	0	0
Chart-5	Ochiai	0	0	1
	Tarantula	0	1	1
	Dstar2	0	1	1
	CNN	0	0	0
Chart-6	Ochiai	0	0	0
	Tarantula	0	0	0
	Dstar2	0	0	0
	CNN	0	0	0
Chart-7	Ochiai	0	0	0
	Tarantula	0	0	0
	Dstar2	0	0	0
	CNN	0	0	0
Chart-8	Ochiai	0	0	0
	Tarantula	0	0	0
	Dstar2	0	0	0
	CNN	0	0	1

PHỤ LỤC

1. Link source code và dataset

https://github.com/nhamct/CNN_FL

2. Thông tin chi tiết các lỗi trong dự án Chart và Closure

<u>Bug id</u>	<u># Files</u>	<u># Classes</u>	<u># Methods</u>	<u># Lines</u>	<u># Added</u>	<u># Removed</u>	<u># Modified</u>	<u># Chunks</u>	<u># Failing tests</u>	<u># Repair Actions</u>	<u># Repair Patterns</u>	First exception
Chart 1	1	1	1	1	0	0	1	1	1	1	2	AssertionFailedError
Chart 2	1	1	2	14	14	0	0	8	2	4	3	NullPointerException
Chart 3	1	1	1	2	2	0	0	1	1	1	2	AssertionFailedError
Chart 4	1	1	1	2	2	0	0	2	22	1	2	NullPointerException
Chart 5	1	1	1	5	4	0	1	2	1	4	2	IndexOutOfBoundsException
Chart 6	1	1	1	8	7	0	1	1	2	8	2	AssertionFailedError
Chart 7	1	1	1	2	0	0	2	2	1	3	3	AssertionFailedError
Chart 8	1	1	1	1	0	0	1	1	1	2	3	AssertionFailedError
Chart 9	1	1	1	1	0	0	1	1	1	1	2	IllegalArgumentException

<u>Bug id</u>	<u># Files</u>	<u># Classes</u>	<u># Methods</u>	<u># Lines</u>	<u># Added</u>	<u># Removed</u>	<u># Modified</u>	<u># Chunks</u>	<u># Failing tests</u>	<u># Repair Actions</u>	<u># Repair Patterns</u>	<u>First exception</u>
Chart 10	1	1	1	1	0	0	1	1	1	2	2	ComparisonFailure
Chart 11	1	1	1	1	0	0	1	1	1	2	3	AssertionFailedError
Chart 12	1	1	1	1	0	0	1	1	1	3	3	AssertionFailedError
Chart 13	1	1	1	1	0	0	1	1	1	6	5	IllegalArgumentException
Chart 14	2	2	4	12	12	0	0	4	4	2	3	NullPointerException
Chart 15	1	1	2	5	5	0	0	3	1	2	4	AssertionFailedError
Chart 16	1	1	2	3	0	0	3	2	8	5	3	NullPointerException
Chart 17	1	1	1	2	1	0	1	1	1	5	2	IllegalArgumentException
Chart 18	2	2	3	13	10	2	1	6	4	9	4	IndexOutOfBoundsException

<u>Bug id</u>	<u># Files</u>	<u># Classes</u>	<u># Methods</u>	<u># Lines</u>	<u># Added</u>	<u># Removed</u>	<u># Modified</u>	<u># Chunks</u>	<u># Failing tests</u>	<u># Repair Actions</u>	<u># Repair Patterns</u>	<u>First exception</u>
Chart 19	1	1	2	6	6	0	0	2	2	3	3	AssertionFailedError
Chart 20	1	1	1	1	0	0	1	1	1	2	3	AssertionFailedError
Chart 21	1	1	2	39	39	0	0	4	1	6	3	AssertionFailedError
Chart 22	1	1	3	33	30	1	2	7	6	11	5	UnknownKeyException
Chart 23	1	1	1	19	19	0	0	1	1	7	2	AssertionFailedError
Chart 24	1	1	1	1	0	0	1	1	1	2	4	IllegalArgumentException
Chart 25	1	1	2	14	12	0	2	6	4	7	6	AssertionFailedError
Chart 26	1	1	1	2	2	0	0	2	22	1	2	AssertionFailedError
Closure 1	1	1	1	3	3	0	0	1	8	2	1	AssertionFailedError

<u>Bug id</u>	<u># Files</u>	<u># Classes</u>	<u># Methods</u>	<u># Lines</u>	<u># Added</u>	<u># Removed</u>	<u># Modified</u>	<u># Chunks</u>	<u># Failing tests</u>	<u># Repair Actions</u>	<u># Repair Patterns</u>	<u>First exception</u>
Closure 2	1	1	1	4	4	0	0	3	1	3	2	NullPointerException
Closure 3	1	2	2	8	6	0	2	3	3	10	2	AssertionFailedError
Closure 4	1	1	1	2	0	0	2	2	3	2	3	AssertionFailedError
Closure 5	1	1	1	3	3	0	0	1	1	3	1	AssertionFailedError
Closure 6	1	1	2	8	0	8	0	4	3	2	2	AssertionFailedError
Closure 7	1	1	1	5	4	0	1	3	2	5	3	AssertionFailedError
Closure 8	1	1	2	4	4	0	0	2	1	4	1	AssertionFailedError
Closure 9	1	2	2	2	1	0	1	2	1	5	2	ComparisonFailure
Closure 10	1	1	1	1	0	0	1	1	1	2	2	AssertionFailedError

<u>Bug id</u>	<u># Files</u>	<u># Classes</u>	<u># Methods</u>	<u># Lines</u>	<u># Added</u>	<u># Removed</u>	<u># Modified</u>	<u># Chunks</u>	<u># Failing tests</u>	<u># Repair Actions</u>	<u># Repair Patterns</u>	<u>First exception</u>
Closure 11	1	1	1	2	0	2	0	1	2	3	1	AssertionFailedError
Closure 12	1	1	1	6	6	0	0	1	1	6	2	AssertionFailedError
Closure 13	1	1	1	2	1	1	0	2	1	1	2	AssertionFailedError
Closure 14	1	1	1	1	0	0	1	1	3	2	3	AssertionFailedError
Closure 15	1	1	1	3	3	0	0	1	1	3	1	AssertionFailedError
Closure 16	1	2	3	9	6	0	3	5	2	8	3	AssertionFailedError
Closure 17	1	1	1	5	4	0	1	2	1	5	2	ComparisonFailure
Closure 18	1	1	1	1	0	0	1	1	1	1	2	AssertionFailedError
Closure 19	1	1	1	2	2	0	0	2	1	1	1	IllegalArgumentException

<u>Bug id</u>	<u># Files</u>	<u># Classes</u>	<u># Methods</u>	<u># Lines</u>	<u># Added</u>	<u># Removed</u>	<u># Modified</u>	<u># Chunks</u>	<u># Failing tests</u>	<u># Repair Actions</u>	<u># Repair Patterns</u>	<u>First exception</u>
Closure 20	1	1	1	2	1	0	1	1	1	2	3	AssertionFailedError
Closure 21	1	1	1	19	0	17	2	2	1	8	4	AssertionFailedError
Closure 22	1	1	1	26	0	24	2	5	1	8	5	AssertionFailedError
Closure 23	1	1	1	7	6	0	1	2	1	5	2	AssertionFailedError
Closure 24	1	1	1	5	2	1	2	4	1	4	3	AssertionFailedError
Closure 25	1	1	1	6	2	4	0	4	1	5	2	ComparisonFailure
Closure 26	1	1	1	5	5	0	0	3	7	5	2	AssertionFailedError
Closure 27	1	1	3	6	3	0	3	3	3	6	3	IllegalStateException
Closure 28	1	1	1	4	4	0	0	1	2	3	2	AssertionFailedError

<u>Bug id</u>	<u># Files</u>	<u># Classes</u>	<u># Methods</u>	<u># Lines</u>	<u># Added</u>	<u># Removed</u>	<u># Modified</u>	<u># Chunks</u>	<u># Failing tests</u>	<u># Repair Actions</u>	<u># Repair Patterns</u>	<u>First exception</u>
Closure 29	1	1	1	10	10	0	0	3	5	6	3	AssertionFailedError
Closure 30	2	3	3	10	8	0	2	6	3	9	6	AssertionFailedError
Closure 31	1	1	1	1	0	1	0	1	1	1	2	AssertionFailedError
Closure 32	1	1	1	18	15	3	0	6	4	8	3	ComparisonFailure
Closure 33	1	1	1	3	3	0	0	1	1	3	1	AssertionFailedError
Closure 34	2	2	3	5	0	2	3	3	1	5	3	StackOverflowError
Closure 35	1	1	1	15	0	13	2	1	1	7	2	AssertionFailedError
Closure 36	1	1	1	3	3	0	0	1	1	3	2	AssertionFailedError
Closure 37	2	2	2	5	4	0	1	3	1	5	2	RuntimeException

<u>Bug id</u>	<u># Files</u>	<u># Classes</u>	<u># Methods</u>	<u># Lines</u>	<u># Added</u>	<u># Removed</u>	<u># Modified</u>	<u># Chunks</u>	<u># Failing tests</u>	<u># Repair Actions</u>	<u># Repair Patterns</u>	<u>First exception</u>
Closure 38	1	1	1	1	0	0	1	1	1	1	2	ComparisonFailure
Closure 39	1	1	1	3	0	0	3	3	2	5	3	ComparisonFailure
Closure 40	1	1	1	3	0	2	1	2	2	2	2	RuntimeException
Closure 41	1	1	2	9	9	0	0	2	3	4	3	AssertionFailedError
Closure 42	1	1	1	7	7	0	0	2	1	3	2	AssertionFailedError
Closure 43	1	1	2	17	17	0	0	4	2	6	5	ComparisonFailure
Closure 44	1	1	1	2	2	0	0	2	1	2	1	ComparisonFailure
Closure 45	1	2	2	6	4	0	2	4	1	8	4	AssertionFailedError
Closure 46	1	1	1	16	0	16	0	1	3	9	3	AssertionFailedError

<u>Bug id</u>	<u># Files</u>	<u># Classes</u>	<u># Methods</u>	<u># Lines</u>	<u># Added</u>	<u># Removed</u>	<u># Modified</u>	<u># Chunks</u>	<u># Failing tests</u>	<u># Repair Actions</u>	<u># Repair Patterns</u>	<u>First exception</u>
Closure 47	2	2	2	8	5	0	3	3	16	5	2	AssertionFailedError
Closure 48	1	1	1	7	3	0	4	2	1	6	2	AssertionFailedError
Closure 49	1	1	3	22	14	8	0	8	66	10	4	AssertionFailedError
Closure 50	1	1	1	6	5	0	1	3	2	3	3	AssertionFailedError
Closure 51	1	1	1	1	0	0	1	1	1	2	2	ComparisonFailure
Closure 52	1	1	1	1	0	0	1	1	1	2	2	ComparisonFailure
Closure 53	1	1	1	4	4	0	0	2	1	4	1	RuntimeException
Closure 54	2	2	3	13	10	1	2	5	3	9	6	ComparisonFailure
Closure 55	1	1	1	2	1	0	1	1	1	2	2	IllegalStateException

<u>Bug id</u>	<u># Files</u>	<u># Classes</u>	<u># Methods</u>	<u># Lines</u>	<u># Added</u>	<u># Removed</u>	<u># Modified</u>	<u># Chunks</u>	<u># Failing tests</u>	<u># Repair Actions</u>	<u># Repair Patterns</u>	<u>First exception</u>
Closure 56	1	1	1	4	4	0	0	2	3	3	1	ComparisonFailure
Closure 57	1	1	1	1	0	0	1	1	1	2	2	AssertionFailedError
Closure 58	1	1	1	4	4	0	0	2	1	2	1	IllegalStateException
Closure 59	1	1	1	2	1	0	1	1	1	2	2	AssertionFailedError
Closure 60	1	1	2	5	5	0	0	3	2	4	2	AssertionFailedError
Closure 61	1	1	1	6	6	0	0	1	3	5	2	AssertionFailedError
Closure 62	1	1	1	1	0	0	1	1	2	1	2	ComparisonFailure
Closure 63	1	1	1	1	0	0	1	1	2	1	2	ComparisonFailure
Closure 64	1	1	3	4	0	0	4	4	1	5	1	AssertionFailedError

<u>Bug id</u>	<u># Files</u>	<u># Classes</u>	<u># Methods</u>	<u># Lines</u>	<u># Added</u>	<u># Removed</u>	<u># Modified</u>	<u># Chunks</u>	<u># Failing tests</u>	<u># Repair Actions</u>	<u># Repair Patterns</u>	<u>First exception</u>
Closure 65	1	1	1	1	0	0	1	1	1	1	2	ComparisonFailure
Closure 66	1	1	1	2	2	0	0	2	2	2	1	AssertionFailedError
Closure 67	1	1	1	1	0	0	1	1	1	2	2	AssertionFailedError
Closure 68	1	1	3	4	3	1	0	4	1	1	4	AssertionFailedError
Closure 69	1	1	1	7	7	0	0	1	3	2	2	AssertionFailedError
Closure 70	1	1	1	1	0	0	1	1	5	1	2	AssertionFailedError
Closure 71	1	1	1	1	0	0	1	1	2	3	3	AssertionFailedError
Closure 72	2	2	2	3	2	0	1	2	1	3	2	AssertionFailedError
Closure 73	1	1	1	1	0	0	1	1	1	1	2	ComparisonFailure

<u>Bug id</u>	<u># Files</u>	<u># Classes</u>	<u># Methods</u>	<u># Lines</u>	<u># Added</u>	<u># Removed</u>	<u># Modified</u>	<u># Chunks</u>	<u># Failing tests</u>	<u># Repair Actions</u>	<u># Repair Patterns</u>	<u>First exception</u>
Closure 74	1	1	2	15	13	0	2	2	3	9	4	AssertionFailedError
Closure 75	1	1	2	4	3	0	1	3	1	5	3	AssertionFailedError
Closure 76	1	1	2	43	37	6	0	11	4	9	5	AssertionFailedError
Closure 77	1	1	1	1	1	0	0	1	1	2	2	ComparisonFailure
Closure 78	1	1	1	2	0	2	0	2	1	1	1	AssertionFailedError
Closure 79	2	2	2	2	1	0	1	2	5	3	2	RuntimeException
Closure 80	1	1	2	2	2	0	0	2	2	1	1	AssertionFailedError
Closure 81	1	1	1	7	7	0	0	1	1	4	1	AssertionFailedError
Closure 82	1	1	1	3	2	0	1	1	2	2	2	AssertionFailedError

<u>Bug id</u>	<u># Files</u>	<u># Classes</u>	<u># Methods</u>	<u># Lines</u>	<u># Added</u>	<u># Removed</u>	<u># Modified</u>	<u># Chunks</u>	<u># Failing tests</u>	<u># Repair Actions</u>	<u># Repair Patterns</u>	<u>First exception</u>
Closure 83	1	1	1	4	3	0	1	1	1	2	1	AssertionFailedError
Closure 84	1	1	3	27	27	0	0	3	1	7	3	AssertionFailedError
Closure 85	1	1	2	21	7	13	1	4	2	9	4	AssertionFailedError
Closure 86	1	1	1	1	0	0	1	1	7	1	3	AssertionFailedError
Closure 87	1	1	1	13	12	0	1	4	1	8	2	AssertionFailedError
Closure 88	1	1	1	6	6	0	0	2	7	5	1	AssertionFailedError
Closure 89	2	2	2	4	3	0	1	2	8	3	3	AssertionFailedError
Closure 90	2	2	2	4	3	0	1	2	2	4	3	AssertionFailedError
Closure 91	1	1	1	9	9	0	0	1	1	5	3	AssertionFailedError

<u>Bug id</u>	<u># Files</u>	<u># Classes</u>	<u># Methods</u>	<u># Lines</u>	<u># Added</u>	<u># Removed</u>	<u># Modified</u>	<u># Chunks</u>	<u># Failing tests</u>	<u># Repair Actions</u>	<u># Repair Patterns</u>	<u>First exception</u>
Closure 92	1	1	1	1	0	0	1	1	1	2	2	AssertionFailedError
Closure 93	1	1	1	1	0	0	1	1	1	2	2	AssertionFailedError
Closure 94	1	1	1	19	19	0	0	3	3	3	2	AssertionFailedError
Closure 95	1	1	1	7	7	0	0	2	2	4	2	AssertionFailedError
Closure 96	1	1	1	4	3	0	1	3	1	3	3	AssertionFailedError
Closure 97	1	1	1	2	1	0	1	1	1	3	3	AssertionFailedError
Closure 98	1	2	2	19	19	0	0	5	1	6	4	AssertionFailedError
Closure 99	1	1	1	6	5	0	1	4	3	6	3	AssertionFailedError
Closure 100	1	1	1	8	7	0	1	2	9	6	3	AssertionFailedError

<u>Bug id</u>	<u># Files</u>	<u># Classes</u>	<u># Methods</u>	<u># Lines</u>	<u># Added</u>	<u># Removed</u>	<u># Modified</u>	<u># Chunks</u>	<u># Failing tests</u>	<u># Repair Actions</u>	<u># Repair Patterns</u>	<u>First exception</u>
Closure 101	1	1	1	4	1	3	0	2	1	2	1	AssertionFailedError
Closure 102	1	1	1	2	1	1	0	2	1	1	2	AssertionFailedError
Closure 103	2	2	2	8	8	0	0	3	3	5	4	AssertionFailedError
Closure 104	1	1	1	1	0	0	1	1	1	2	2	AssertionFailedError
Closure 105	1	1	1	7	2	0	5	5	1	7	4	AssertionFailedError
Closure 106	2	2	2	6	4	2	0	3	4	4	3	AssertionFailedError
Closure 107	1	1	1	1	1	0	0	1	1	1	2	AssertionFailedError
Closure 108	1	1	3	4	3	0	1	4	1	4	3	IllegalStateException
Closure 109	1	1	1	5	4	0	1	1	2	5	4	AssertionFailedError

<u>Bug id</u>	<u># Files</u>	<u># Classes</u>	<u># Methods</u>	<u># Lines</u>	<u># Added</u>	<u># Removed</u>	<u># Modified</u>	<u># Chunks</u>	<u># Failing tests</u>	<u># Repair Actions</u>	<u># Repair Patterns</u>	<u>First exception</u>
Closure 110	2	2	2	25	21	0	4	12	2	12	8	AssertionFailedError
Closure 111	1	1	1	2	1	0	1	1	1	3	2	AssertionFailedError
Closure 112	1	1	1	9	6	0	3	1	2	6	1	AssertionFailedError
Closure 113	1	1	1	1	0	0	1	1	1	2	2	AssertionFailedError
Closure 114	1	1	1	1	0	0	1	1	1	2	2	AssertionFailedError
Closure 115	1	1	1	11	0	11	0	2	7	5	2	AssertionFailedError
Closure 116	1	1	2	12	12	0	0	2	8	5	3	AssertionFailedError
Closure 117	1	1	1	24	12	12	0	3	1	9	1	ComparisonFailure
Closure 118	1	1	1	3	3	0	0	1	2	2	1	ComparisonFailure

<u>Bug id</u>	<u># Files</u>	<u># Classes</u>	<u># Methods</u>	<u># Lines</u>	<u># Added</u>	<u># Removed</u>	<u># Modified</u>	<u># Chunks</u>	<u># Failing tests</u>	<u># Repair Actions</u>	<u># Repair Patterns</u>	<u>First exception</u>
Closure 119	1	1	1	1	1	0	0	1	1	1	2	AssertionFailedError
Closure 120	1	1	1	3	3	0	0	1	1	3	1	AssertionFailedError
Closure 121	1	1	1	3	2	0	1	2	1	2	2	AssertionFailedError
Closure 122	1	1	1	2	1	0	1	1	3	5	2	AssertionFailedError
Closure 123	1	1	1	1	0	0	1	1	1	3	4	ComparisonFailure
Closure 124	1	1	1	2	2	0	0	2	1	2	2	AssertionFailedError
Closure 125	1	1	1	1	0	0	1	1	1	2	2	IllegalStateException
Closure 126	1	1	1	4	0	4	0	1	2	4	1	AssertionFailedError
Closure 127	1	1	2	10	9	0	1	2	6	5	4	AssertionFailedError

<u>Bug id</u>	<u># Files</u>	<u># Classes</u>	<u># Methods</u>	<u># Lines</u>	<u># Added</u>	<u># Removed</u>	<u># Modified</u>	<u># Chunks</u>	<u># Failing tests</u>	<u># Repair Actions</u>	<u># Repair Patterns</u>	First exception
Closure 128	1	1	1	4	3	0	1	2	1	3	2	ComparisonFailure
Closure 129	1	1	1	3	3	0	0	1	1	3	2	AssertionFailedError
Closure 130	1	1	1	1	0	0	1	1	1	1	2	AssertionFailedError
Closure 131	1	1	1	2	1	0	1	2	2	2	2	AssertionFailedError
Closure 132	1	1	1	3	2	0	1	1	1	2	2	AssertionFailedError
Closure 133	1	1	1	1	1	0	0	1	1	1	2	IllegalStateException

TÀI LIỆU THAM KHẢO

- [1] M. Ernst, “Static and dynamic analysis: synergy and duality,” 2003.
- [2] J. S. C. a. L. Cousins, “Towards automatic software fault location”.
- [3] M. R. G. S. K. e. a. Harrold, “An empirical investigation of the relationship between spectra differences and regression faults.,” *Journal of Software Testing, Verification and Reliability* , 2000.
- [4] G. R. K. S. R. W. a. L. Y. M. J. Harrold, “An empirical investigation of the relationship between spectra differences and regression faults”.
- [5] M. C. a. F. K. H. De Souza, “Spectrum-based Software Fault Localization: A Survey of Techniques, Advances, and Challenges”.
- [6] J. A. J. a. M. J. Harrold, “Empirical evaluation of the tarantula automatic,” trong *International Conference on Automated*, 2005.
- [7] P. Z. a. A. J. v. G. R. Abreu, “An Evaluation of Similarity Coefficients for Software Fault Localization,” trong *Proceedings of the 12th Pacific Rim International Symposium on Dependable Computing*, 2006.
- [8] D. L. L. J. a. A. B. Lucia, “Comprehensive Evaluation of Association Measures for Fault Localization,” trong *International Conference on Software Maintenance*, 2010.
- [9] M. N. X. A. Z. A. A. a. I. M. J. B. Liblit, “Scalable statistical bug isolation,” trong *PLDI*, 2005.
- [10] X. Y. L. F. J. H. a. P. S. M. C. Liu, “Sober: Statistical model based bug localization,” SIGSOFT, Softw. Eng. Notes, 2005.
- [11] “Defects4j dataset,” [Trực tuyến]. Available: <https://github.com/speezepearson/defects4j>.