情報処理学会論文誌 Vol.59 No.4 1250–1261 (Apr. 2018)

深層学習による ソースコードコミットからの不具合混入予測

近藤 将成[1,2,a)] 森 啓太[1] 水野 修[1,b)] 崔 銀惠[2,c)]

受付日 2017年8月1日, 採録日 2018年1月15日

概要：ソフトウェアの不具合予測は，ソフトウェアに潜む不具合を予測することで効率的なレビューやテストを可能にしようとするソフトウェア品質保証活動の１つである．従来の多くのソフトウェアの不具合予測では，ソースコード分析による不具合予測を行っているが，粒度が粗くまた不具合予測の結果のフィードバックが遅い．この問題を解決するために，ソフトウェアの変更がコミットされたときに，その変更によって不具合が起きるかどうかを予測する手法が提案され，近年注目を集めている．ソフトウェアの変更コミットの不具合予測に関する既存研究では，その変更に対するメトリクス（たとえば，修正されたファイル数，追加されたコード行数など）を計算した後に機械学習や深層学習を適用している．それに対して，本研究では，変更のソースコード片のみに対して深層学習を適用することで不具合を予測する手法，Word-Convolutional Neural Network（W-CNN）を提案する．我々は，評価実験によって，変更ソースコード片に対する深層学習を用いた不具合予測が可能であること，さらに，提案手法 W-CNN は先行研究に比べて，学習の時間はかかるものの，不具合予測の精度が優れており，予測時間が短いことを示す．

キーワード：不具合予測，畳み込みニューラルネットワーク，変更ソースコード，ソースコード片，文脈

Just-in-Time Defect Prediction Applying Deep Learning to Source Code Changes

Masanari Kondo[1,2,a)] Keita Mori[1] Osamu Mizuno[1,b)] Eun-Hye Choi[2,c)]

Received: August 1, 2017, Accepted: January 15, 2018

Abstract: Defect prediction is an important task for preserving software quality. A lot of previous research has analyzed source code to predict defects; however, it contains a problem that is its prediction grain is too coarse and its feedback is too late for software developers. To achieve a more fine-grained prediction and an earlier feedback, several approaches that analyzes *source code changes* has been reported. Those approaches have applied various machine learning techniques and deep learning techniques to change metrics, such as the number of lines added, modified files, and modified directories. In this paper, we propose a novel approach for defect prediction called *Word-Convolutional Neural Network* (*W-CNN*), which applies CNN to the modified source code itself. Our evaluation results show that the proposed approach can improve the effectiveness of defect prediction with a small overhead on the prediction time.

Keywords: defect prediction, convolutional neural network, source code changes, source code snippet, context

1. はじめに

ソフトウェアの品質は現在の我々の生活に大きな影響を与えており，テストやレビューといった品質保証活動が重要である．ソフトウェアの品質保証を効率的に行うための１つの手法として，不具合予測手法の研究がなされてきてい

[1] 京都工芸繊維大学
Kyoto Institute of Technology, Kyoto 606–8585, Japan
[2] 産業技術総合研究所
National Institute of Advanced Industrial Science and Technology, Osaka 563–8577, Japan
[a)] m-kondo@se.is.kit.ac.jp
[b)] o-mizuno@kit.ac.jp
[c)] e.choi@aist.go.jp

© 2018 Information Processing Society of Japan

る [2], [7], [10], [17], [24], [25], [27]．不具合予測とは，過去のソフトウェア開発の履歴情報やソースコードから取り出された特徴を用いて，不具合が混入しそうなソフトウェアのモジュールやファイルを予測する手法である．この予測結果を用いることで，効率的なテストやレビューを行い，リリース後のソフトウェアの不具合を減らすことが可能である．

近年，ツールの発展 [1], [6], [18] や潤沢でオープンアクセス可能なデータの公開 [9], [15], [20] により，機械学習を用いた不具合予測手法がさかんに研究されている [2], [7], [10], [24], [25], [27]．中でも，深層学習の他分野での発展 [4], [11], [13], [26] を受けて，深層学習を用いた不具合予測手法の研究にも注目が集まっている [24], [25]．

深層学習を用いた不具合予測の既存研究 [24], [25] では，深層学習アルゴリズムの 1 つであるディープビリーフネットワークを用いることで，従来の特徴セットから不具合予測に対して有効な特徴セットを生成できることを示している．Yang らの研究 [25] では，変更に対するソフトウェアメトリクスに対して深層学習を適用している．また，Wang らの研究 [24] では，ソースコードを抽象構文木によって解析し，その特徴を用いている．

しかし，我々が知る限り，変更のソースコード片に対して深層学習を適用した不具合予測手法の研究はいまだ行われていない．変更に対する不具合予測は，ソフトウェアの変更コミット時に細かい粒度で不具合を予測でき，予測結果のフィードバックも速い，という利点がある [9]．また，ソースコードに対する不具合予測手法では，メトリクスを使用せずにテキストのみを利用して予測する手法が示されており [17]，同様に変更のソースコード片に対しても，メトリクスを用いずに予測することが可能である [2], [7], [10]．

そこで，本研究では，変更のソースコード片に対して深層学習を適用して不具合予測を行う手法を提案し，提案法の不具合予測性能を評価する．深層学習には，畳み込みニューラルネットワーク（Convolutional Neural Network: CNN）を用いる．CNN は主に画像のカテゴリ分類での成功が大きいが [13]，近年ではテキスト分類でも成果を示している [4], [11], [26]．我々はソースコード片をテキストと見なし，不具合を起こすか否かの分類に CNN を適用する．適用する CNN モデルは，Kim のモデル [11] を参考にして構築する．このモデルは，テキストの単語レベルの情報を用いて分類を行う．以上の特徴を備えた我々の提案手法を，W-CNN（Word-CNN）と呼ぶこととする．

本研究では，提案手法 W-CNN を評価するために以下の 3 つの研究設問を設定する．

RQ1：W-CNN による不具合の学習・予測は可能か？

RQ2：W-CNN は既存の変更に対する深層学習を用いた不具合予測手法よりも予測精度が良いか？

RQ3：W-CNN の学習・予測時間はどの程度か？

これらの研究設問に応えるために，我々は，提案手法の W-CNN と，ソースコードの変更メトリクスに対して深層学習を適用する既存の不具合予測手法である Yang ら [25] の手法（Deeper）との比較評価実験を行う．実験では，7 つの Java もしくは C++で書かれたオープンソースソフトウェアプロジェクトを用いて，予測精度と学習時間を調査する．その結果，RQ1 に対して，提案手法 W-CNN は対象データに対する不具合の学習，および，不具合の予測が可能であることが分かった．RQ2 に対して，W-CNN は既存の深層学習を用いた不具合予測手法 Deeper と比較して予測精度を計る一般的な尺度 AUC（Area Under the receiver operating characteristic Curve）の値を平均 22%程度上げることができた．RQ3 に対して，W-CNN は既存の手法 Deeper より学習に時間がかかるが，学習後の各コミットに対する予測時間は 0.0004 秒程度と非常に短く，変更数に対する実行時間のオーバヘッドは少ないことが分かった．

本研究の主な貢献を以下にまとめる．

(1) ソフトウェアの変更コミット時のソースコード片に対して，深層学習を適用することで，不具合予測が可能であることを，我々が知る限り初めて示した．
(2) 提案手法 W-CNN は，既存の不具合予測手法よりも学習に時間がかかるが，高い不具合予測精度をもたらすことが可能であることを確認した．

以降の本論文の構成を紹介する．2 章では，不具合予測，および，深層学習を適用した不具合予測の関連研究について述べる．3 章では，提案手法である W-CNN について説明する．4 章では，評価実験について説明する．5 章では，それぞれの研究設問に対する結果をまとめる．6 章では，本研究の妥当性の検証を行う．7 章では，本研究の結論を述べる．

2. 関連研究

2.1 不具合予測手法

不具合予測はソフトウェアの品質保証活動の 1 つとして重要な分野であり，これまでに様々な不具合予測手法が提案されてきている [2], [7], [10], [17], [24], [25], [27]．たとえば，Zimmermann らの研究 [27] では，ソースコードに関する複雑度メトリクスを用いてロジスティック回帰モデルを作成し不具合の予測を行っている．しかし，このように複雑度メトリクスなどを用いてファイルやパッケージレベルで不具合を予測する手法にはいくつかの問題が指摘されている [9]．たとえば，予測する時期が遅いという問題がある．開発者は，ソースコードを書いているときに，その追加した機能に不具合が含まれているかを知りたい．しかし，ファイルやパッケージレベルではこの段階での予測が難しい．そのため，追加した機能の記憶が薄れてから修正を行うことになり，効率的であるとはいいにくい．

この問題を解決するために，Kamei らの研究 [9] では，ソフトウェアの変更に対するメトリクス（以降，変更メトリクス）を用いる手法を提案している．開発者が変更（コ

© 2018 Information Processing Society of Japan

ミット）を行った際に，その変更に不具合が含まれているかどうかを判断する方法により，変更を行ったすぐ後に不具合が含まれていそうか否かが分かるため，素早いフィードバックや，品質保証活動を適切な開発者に割り当てられるなどの利点が期待できる [8]．よって，今回の我々の研究では，変更に対する不具合予測手法を提案する．

その他の不具合予測手法としては，ソースコードなどのテキストに注目している手法も研究されている [2], [7], [10], [17]．テキストは複雑度メトリクスなどと比較して収集が容易であるという利点がある [17]．テキストとしてはソースコードが用いられることが多く，テキスト分類技術によって不具合が混入しているテキストとそうではないテキストを分類する手法が研究されている．たとえば Mizuno らの研究 [17] では，ソースコードのみを学習データとし，スパムフィルタを用いて学習することで不具合予測を行っている．Mizuno らの研究では，ソフトウェアが更新される時系列順を用いた適用実験も行っており，学習数が少ない間は低い精度であったのが，多くの学習を繰り返すと精度が向上することを確認している．ただし，時系列を考慮した実験では実践的な結果が得られる反面，現実の開発環境を模すための設定事項が多くなるため，これまでにあまり試みられてはいない．また，提案手法に似た手法として，変更に関するテキスト情報を用いた不具合予測手法がいくつか提案されているが [2], [7], [10]，変更のソースコード片に対して深層学習を用いた不具合予測手法は提案されていない．

2.2 深層学習を用いた不具合予測手法

近年，不具合予測のために深層学習を適用する研究が行われている．Yang らの研究 [25] では，亀井らの提案している変更メトリクス [9] から特徴を生成し，その特徴を入力としてディープビリーフネットワークを用いて不具合予測を行っている．Wang らの研究 [24] は，ソースコードを抽象構文木で分析し，その特徴をディープビリーフネットワークで分析し，ソースコードの意味的な違いを表現することで，不具合予測の精度を向上できることを示している．Lam らの研究 [14] は，バグレポートからの不具合推定を行うために深層学習と IR 技術を結合する手法を提案し，既存の手法よりも推定精度が良くなることを示している．しかし，これらの研究では，変更に関するテキスト情報に対して深層学習を適用していない．それに対して我々は，変更ソースコードのみに対して CNN の深層学習のみを適用するシンプルなテキストベースの不具合予測手法を提案する．

提案手法との比較のために，我々と同様に変更に対する不具合予測手法であり，かつ深層学習を用いている Yang らの Deeper [25] を選択する．提案手法 W-CNN と Deeper の大きな違いは以下の 3 点である．

(1) 深層学習のモデル：W-CNN は CNN を用いるが，Deeper はディープビリーフネットワークを用いている．
(2) 分類手法：W-CNN は深層学習を用いるが，Deeper はロジスティック回帰を用いている．
(3) 分析するデータ対象：W-CNN はソースコード片を用いるが，Deeper はメトリクスを用いている．

3. 提案手法

深層学習の一種である畳み込みニューラルネットワーク（CNN）は人の網膜を模倣したニューラルネットワークであり，画像分類で成功を収めてきたアルゴリズムである [13]．最近ではテキスト分類においても高い精度が得られることが分かってきている [4], [11], [26]．我々は不具合予測問題を，ソースコードを特徴として用いるテキスト分類問題としてとらえ，CNN を用いた解法を提案する．

一般に深層学習は層の数が多い程表現力が良くなる反面，大量の学習データと時間を要する傾向にあるため，大規模な深層学習はソフトウェアプロジェクトに適用することが難しい．たとえば，Zhang らの畳み込みニューラルネットワークを用いたテキスト分類モデル [26] では，隠れ層が 9 層の CNN を構成しており，100 万個程度の訓練データを用いている．しかし，我々が対象とするソフトウェアプロジェクトの不具合予測に用いる変更の学習データ数は，比較的に大きい場合でも数千から 1 万個程度である．

そのため，我々は Kim により提案されている比較的に小規模なテキスト分類モデル [11] を利用する．このモデルは隠れ層が 3 層の CNN であり，Zhang らの 9 層の CNN と比較して軽量である．また，Kim のモデルは 4,000 から 10,000 程度の訓練データでネットワークを学習可能であることから，ソフトウェアプロジェクトの不具合予測に対しても適用可能であると判断する．我々は Kim のモデルを参考に，ソースコードの単語レベルの情報を用いた分類機を作成する．

図 1 に提案する不具合予測手法 W-CNN の構成の概要を示す．提案手法は，分析対象のリポジトリからソースコード片を取り出す前処理（Preprocess）と，ソースコード片から特徴を取り出し分類の処理を行うネットワーク（Network Architecture）からなる．処理の流れを以下に説明する．

(1) 前処理：分析対象のリポジトリから得られた各コミットの追加・修正されたソースコードを単語分割し，それぞれのコミットの特徴とする．
(2) ネットワーク：コミットの単語分割されたソースコード片を入力とし，ネットワークで特徴の学習，および，分類を行う．

3.1 前処理

(1) ソースコード片の作成：ソースコード片は，変更に

© 2018 Information Processing Society of Japan

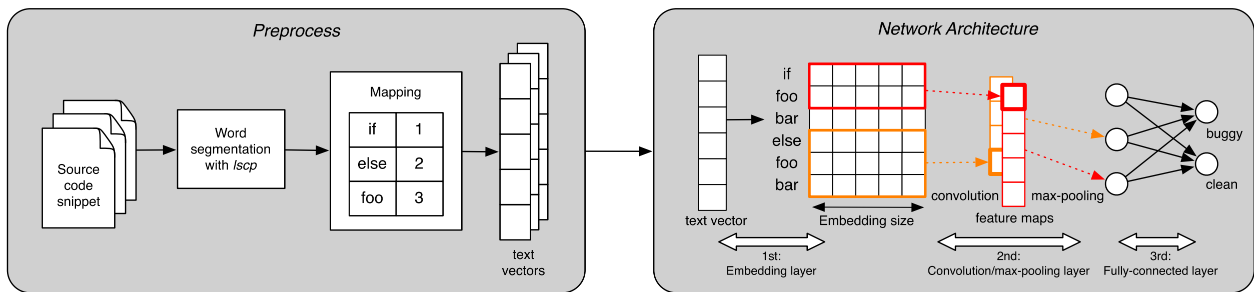


図 1　提案手法 W-CNN の構成の概要
Fig. 1　The outline of proposed defect prediction W-CNN.

より追加・修正された差分のソースコードをつなぎ合わせた文字列である．作成方法について述べる．取得する変更の差分はソースファイル*1のみからとする．これは，ドキュメントファイルなどは不具合につながる可能性が低いと考えられるためである．不具合を混入したコミットを見つけることを目的とするため，対象のソースコードとして追加もしくは修正された行を集める（ただし，削除行は含めない．理由は後述する）．さらに，追加修正行の前後 3 行を文脈情報として集める．これは，変更におけるコード片の情報のみを使う場合よりも，その前後のソースコードの文脈情報を追加する方が，より自然言語処理のようにソースコードの意味を学習可能であると仮説を立てたためである．我々が知る限り，文脈行を不具合予測に適用した先行研究はこれまでにない．そこで，Hadoop プロジェクトに対して，0 行から 10 行までの文脈行数の不具合予測への影響を測定した．Git の diff コマンドで標準で出力される前後 3 行が最も良かったため，文脈行数は 3 とする（**図 2**）．より詳しくは妥当性の検証の構成概念妥当性で述べる．ただし，コメント行はソフトウェアの動作に関係がないため取得しない．以上のソースコードを，変更されたファイルごとに取得し，それぞれのソースコードの先頭にそのソースコードを取得したファイル名を付与した文字列を連結することで 1 つのソースコード片とする．ソースコードのファイル名を付与する理由は，そのソースコード片がどのような機能を持つソースコードの一部であるのかを，最も端的に示すことができる文字列だからである．変更のソースコード片の例を**図 3** に示す．なお，ここでは 2 つのコミットからソースコード片の一部を抜粋している．赤で示されているのがファイル名，それ以外がソースコード片である．

提案法では，削除行そのものはソースコード片として考慮しない．ただし，削除行の文脈行は考慮する．これは，削除行そのものは，そのコミットを取得した際に削除されるため，直接の不具合の原因にはならないと考えられるためである．ただし，削除することによりその前後のソースコードに影響を与え不具合の原因となることは考えられるため，その文脈行はソースコード片として考慮する．

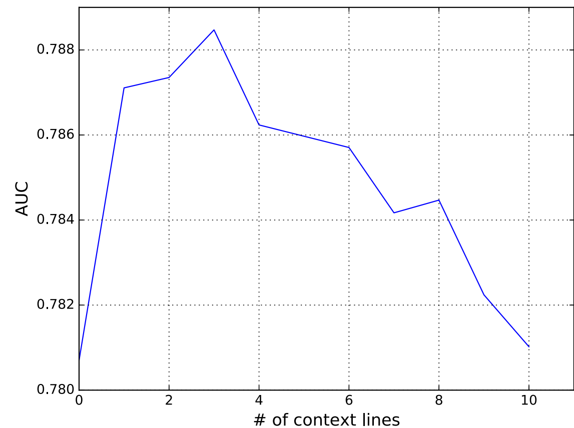


図 2　文脈行数（# of context lines）と W-CNN による分類精度 AUC の関係（Hadoop Project の場合）
Fig. 2　A relationship between # of context lines and AUC on prediction accuracy by W-CNN (Hadoop).

[Commit: d80f93cca26f82126f408179fdc8c3c6c1ccbc7f のソースコード片]

components/camel-kafka/src/main/java/org/apache/camel/component/kafka/KafkaConfiguration.java } public String getSaslMechanism() { return saslMechanism; } public void setSaslMechanism(String saslMechanism) { this.saslMechanism = saslMechanism; } public String getSecurityProtocol() { return securityProtocol; }

[Commit: 2645cc184f549da4c2ce398a8ea9704927524b2e のソースコード片より一部抜粋]

components/camel-kafka/src/main/java/org/apache/camel/component/kafka/KafkaConfiguration.java private Integer reconnectBackoffMs = 50; @UriParam(label = "common", defaultValue = SaslConfigs.DEFAULT_SASL_MECHANISM) private String saslMechanism = SaslConfigs.DEFAULT_SASL_MECHANISM; @UriParam(label = "common", defaultValue = SaslConfigs.DEFAULT_KERBEROS_KINIT_CMD) private String kerberosInitCmd = SaslConfigs.DEFAULT_KERBEROS_KINIT_CMD; @UriParam(label = "common", defaultValue = "60000") private Integer kerberosBeforeReloginMinTime = 60000; @UriParam(label = "common", defaultValue = "0.05") private Double kerberosRenewJitter = SaslConfigs.DEFAULT_KERBEROS_TICKET_RENEW_JITTER;

図 3　変更のソースコード片の例（Camel Project の場合．2 つのコミットから一部を抜粋）
Fig. 3　An example of a code snippet in a change level.

（2）単語分割：Kim のモデルでは，入力文書を単語分割しているため，我々もソースコード片を単語分割する必要

*1　ここで，ソースファイルとは，分析対象が C++，および，Java であったことから，拡張子が java，c，h，cpp，hpp，cxx，hxx のファイルとする．

© 2018 Information Processing Society of Japan

がある．ソースコード片の単語分割には Thomas が公開している lscp（A lightweight source code preprocessor）[23] を用いる．lscp はソースコードを構文解析せずに経験則的に単語分割を行うためソースコード片にも適用可能である．具体的には，入力を正規表現によってコメントとそれ以外の部分に分類し，記号の除去などによって，プログラミング言語の予約語や識別子のみを抽出している．lscp は多くのパラメータにより動作を規定することができ，本研究で用いたパラメータは以下のとおりである．

- コードを対象とする．
- 識別子を抽出する．
- コメントを除去する．
- 数値を除去する．
- 単語を小文字化する．
- 語幹抽出をしない．
- 合成語を分割する．
- 記号（ピリオド，コンマ，疑問符，括弧など）を除去する．
- 短い語（1 文字以下）を除去する．
- ストップワードを除去しない．
- Java，C，C++などの言語特有のキーワードを除去しない．

この結果，ソースコード片の演算子や括弧などはすべて削除され，識別子や予約語のみの列が得られる．

CNN は固定長の入力のみを受け付けるという制約があるため，ソースコード片の単語数を固定長にする必要がある．本実験では，2,000 語を閾値として，それ以上の場合は切り詰め，それ未満の場合は足りない文字数を 0 で補う 0 埋めを行って，固定長の入力を生成する．たとえば，2,003 語を持つソースコード片は，先頭から 2,000 語までを用いて，残りの 3 語を切り捨てる．一方で，1,800 語のコミットは，2,000 語として処理するために足りない残りの 200 語を 0 として処理する．つまり，全コミットのソースコード片の単語数がすべて 2,000 語で統一されることになる．0 埋めは深層学習を用いる際に一般に用いられる手法であるが，切り詰めの処理は一般には行われない．今回の場合，最も多くの単語数を持つコミットに合わせて固定長データを作成すると，入力が非常に大きくなり計算が難しくなってしまう．そこで，閾値として適切な値を調べた結果，多くのプロジェクトで 90%以上のコミットが 2,000 語以下で収まることを確認したため，提案法では 2,000 を閾値として切り詰めの処理を行う．切り詰め方法としては，先頭から数えて 2,000 語以降を切り捨てる簡潔な方法を採用する．

（3）マッピングとベクトル変換：最後に，手に入れたトレーニングデータのすべてのコミットの 2,000 語の単語に対して頻出単語順に 1 から数字をマッピングし，トレーニングデータ全体で 1 つのマッピングテーブルを作成する．マッピングテーブルを利用して，訓練データ，および，テストデータをテキストベクトル（Text vectors）へ変換する．ただし，マッピングテーブルにない単語は 0 とする．ここでマッピングテーブルにないとは，トレーニングデータのどのコミットの 2,000 語の単語の中にも含まれていない単語を指す．ここでの出力は，ソースコード片を持つコミットが 1 次元の数値ベクトルに変換される．数値ベクトルの各値はそのコミットのソースコード片の各単語に対応する．

数値へのマッピングは，CNN が数値ベクトルのみを処理可能であるため，必要である．マッピングの際，頻出単語順に 1 から数字をマッピングする理由は，数値の大きさによって単語の重要度に差をつけるためである．たとえば，画像の認識では，数値の大きさは画素の色に対応しており，その数値の差をエッジとしてとらえるなどが行われるが，提案法ではそれを模倣している．なお，提案法では，マッピングテーブルにない単語は重要単語ではないと仮定し，0 へマッピングすることでその影響をなくす．

3.2 採用するネットワーク

前処理によって得られるテキストベクトルを各コミットの特徴とし，W-CNN のネットワークに入力する．ネットワークでは，特徴の抽出，および，分類が CNN によって行われる．この CNN の実装は，Google が公開している Tensorflow [1] を用いる．これは，実装例やチュートリアルが豊富であり，また，我々の実験環境では，他のライブラリと比較して実行速度が高速であるためである．以下，分類に用いられる 3 層のネットワークについて説明する．

第 1 層は埋め込み表現層（Embedding layer）で，lscp で分割したそれぞれの単語（作成されたテキストベクトルの各値）の埋め込み表現を学習する．埋め込み表現とは各単語を表現するベクトルを求めることであり，Mikolov ら [16] や Pennington ら [19] によって提案されている．Kim のモデル [11] では埋め込み表現学習ツール word2vec [16] を利用しているが，word2vec の学習には大量のデータが必要である．Kim のモデルでは Google News のデータを用いることで 1,000 億の単語を持つコーパスを利用しているが，我々の分類対象はソースコード片であり，Google News のような自然言語のコーパスでは良い識別結果を得られなかった．また，トレーニングデータのコード片を word2vec の学習に用いる実験も行ったが，トレーニングデータ不足により，こちらも良い識別結果を得られなかった．そのため，本研究では埋め込み表現を CNN のネットワークに組み込み，CNN の一部として学習させる．この層では単語を 128 次元のベクトルへと変換するためのルックアップテーブルを作成する．この 128 が埋め込み表現のベクトルサイズ（Embedding size）である．Kim のモデルでは 300 次元の埋め込み表現を採用している．しかし，計算時間が長くなるため，今回はサイズを縮小する．一般的に 2 の階乗のサイズがよく使われるため，256 もしくは 128 を検討し，

© 2018 Information Processing Society of Japan

256 では 300 との大幅な差が期待できなかったため，その 1 つ下の 128 次元を利用する．

第 2 層は畳み込み・プーリング層（Convolution/max-pooling layer）である．畳み込み（Convolution）では，新しい特徴を抽出するためのフィルタを学習する．本手法では n-gram [3] のように複数の単語の共起を加味するため，複数の単語ベクトルを同時に処理できるようなフィルタを適用する．ここで用いるフィルタは，横幅が埋め込み表現のベクトルサイズの 128 で高さが 3，4，5 の 3 種類のフィルタ各 128 個である．つまり，3，4，5 単語の共起を考慮している．それぞれ 128 個のフィルタを持つ 3 種類のフィルタをかけるため，フィルタ適用の結果として計 384 個の特徴マップ（Feature maps）が得られる．フィルタの高さは，Kim のモデルと同じ設定にしている．フィルタの個数は，Kim のモデルにおけるフィルタ数よりも数を増やし，2 の階乗となるようにしている．

次に，プーリングでは，フィルタから得られた特徴マップから重要な特徴を抽出するためにマックスプーリング（Max-pooling）を適用する．ここでのマックスプーリングは各フィルタから得られた特徴マップの最大値を最も重要な情報として保存する手法である．そのため，畳み込みによりフィルタが適用された後の 384 個の特徴マップを 384 個のノードに変換することができる．この設定は Kim のモデルと同じである．

第 3 層は全結合層であり，出力のノードを 2 つとして，不具合である確率をソフトマックス関数を用いて計算する．

3.3　ハイパーパラメータと訓練データ

本研究におけるハイパーパラメータは，Kim のモデルを参考にして設定している．ただし，学習におけるパラメータの更新の際の最適化アルゴリズムには Adam [12] を利用する．Adam はニューラルネットワークの学習でよく利用されている確率的勾配降下法（SGD）よりも早く誤差を収束させることができるアルゴリズムであると報告されている．そのため，より高速な学習が可能と考えたためである．また，Kim のモデルは正規化項として CNN の重みに上限を設ける手法を採用しているが，この正規化では良い識別結果が得られなかった．そのため，より単純な 2 乗ノルムを採用する．また，ミニバッチサイズを 64 とし，64 データずつパラメータの更新を行う．

4.　評価実験

提案手法の評価実験では，Yang らの Deeper [25] と比較評価を行う．評価実験の概要を図 4 に示す．実験の主要なステップを以下にまとめる．

(1) 提案手法に関しては，リポジトリから変更（コミット）を取得し，Commit Guru [9], [20] を用いて，変更が不具合を混入したかどうかのラベル情報を基に変更にラベル付けをする．各変更に対し，変更されたソースコードを収集し変更のソースコード片を作成する．
(2) Deeper に関しては，Commit Guru から 14 個の変更メトリクスを取得する．
(3) どちらに対しても同じリサンプリングを行う．
(4) 提案手法 W-CNN と比較対象の不具合予測手法 Deeper に対して 10×10 重交差検証による予測精度の評価を行いその結果をまとめる．

以降，それぞれの方法について詳細を説明する．

4.1　Commit Guru による分析データの準備

不具合予測手法の評価において実験データの公開性と透明性は重要である [8]．そのため，我々は Rosen らが公開している Commit Guru [9], [20] から得られるデータを利用する．Commit Guru は Git のリポジトリを指定すると変更メトリクス [9]，および，不具合混入コミットの情報を自動で計算可能な Web アプリケーションである．

本研究では，不具合混入コミットの情報をコミットのラベル付けに利用する．また，メトリクスを比較評価のための不具合予測手法 Deeper の入力で用いる．Commit Guru が行う（1）不具合混入コミットのラベル付けと（2）メトリクスの収集方法について以下に述べる．

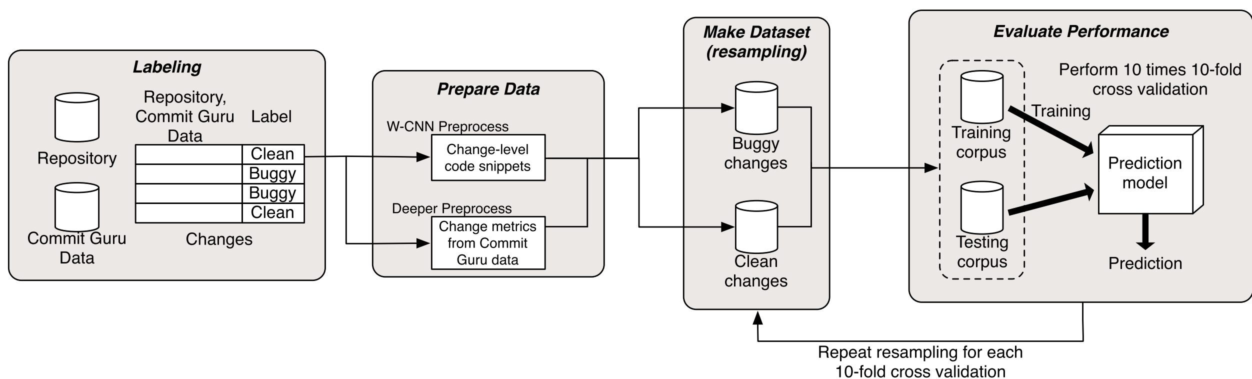


図 4　評価実験の概要
Fig. 4　The outline of experiments.

© 2018 Information Processing Society of Japan

表 1 変更メトリクス
Table 1 Change metrics.

分類	名称	定義
Diffusion	NS	修正されたサブシステムの数
	ND	修正されたディレクトリの数
	NF	修正されたファイルの数
	Entropy	修正されたコードの各ファイルごとの分布
Size	LA	追加されたコード行数
	LD	削除されたコード行数
	LT	変更される前のファイルのコード行数
Purpose	FIX	バグ修正の変更か否か
History	NDEV	修正に関わった開発者の人数
	AGE	最後の変更から最新の変更までの平均時間
	NUC	ユニークな変更の数
Experience	EXP	開発者の経験
	REXP	最近の開発者の経験
	SEXP	サブシステムについての開発者の経験

(1) 不具合混入コミットのラベル付け：Commit Guru はコミットに対し，不具合が混入したコミット（buggy）とそれ以外のコミット（clean）にラベル付けをする．不具合混入コミットは不具合を修正したコミットから推定する．Commit Guru は，まずコミットメッセージを解析し，Hindle らの研究 [5] におけるコミットを分類するためのキーワード（たとえば bug や fix など）があれば，そのコミットが不具合を修正しているコミットであると決める．次に，その修正を行ったコミットの修正行を特定し，その行が追加されたコミットを不具合が混入したコミットとしてラベル付けする．他の不具合コミットを推定する一般的な手法として SZZ アルゴリズム [21] があるが，公開されている信頼性の高いライブラリなどがなく，それぞれの研究で独自の実装が試みられているため，公開性という観点で SZZ アルゴリズムではなく Commit Guru を利用する．

(2) メトリクスの収集：収集されるメトリクスは亀井らの 14 個の変更メトリクス [9] である（**表 1**）．これらのメトリクスを，比較評価のための不具合予測手法 Deeper の入力として用いる．

4.2 対象とするプロジェクトデータ

本研究では，不具合予測手法を作成するために十分な履歴のある 7 つのオープンソースソフトウェアプロジェクト（Hadoop，Camel，Gerrit，Osmand，CMake．Bitcoin，Gimp）のリポジトリを利用する．また，プロジェクトは，それぞれすでに Commit Guru で解析され，14 個の変更メトリクスと不具合混入コミットかどうかのラベルが利用可能である．対象プロジェクトの詳細を**表 2** に示す．

表 2 対象プロジェクト
Table 2 Subject datasets.

Project	Language	The Total Number of Changes	Buggy Rate
Hadoop	Java	13,920	24.8%
Camel	Java	24,740	23.2%
Gerrit	Java	18,794	20.1%
Osmand	Java	31,366	14.0%
CMake	C++	28,400	10.1%
Bitcoin	C++	11,093	14.4%
Gimp	C++	37,116	22.5%

なお，Commit Guru で分析可能なデータは Git リポジトリで管理されているデータであるが，比較対象の先行研究 [25] で分析されているほとんどのプロジェクトのソースコード管理システムが Git ではないため，先行研究と同じプロジェクトのセットを使用することはできなかった．我々は，Commit Guru を利用可能なプロジェクトの中から様々な分野（サーバやアプリ）のプログラムでかつ異なる言語で書かれているプロジェクトを利用する．

4.3 比較手法 Deeper

比較手法の Deeper [25] では，ディープビリーフネットワークが採用されている．この論文から得られた Yang らが採用しているディープビリーフネットワークについての情報は，ネットワークのアーキテクチャ，および，メトリクスの前処理についてである．Deeper のディープビリーフネットワークは以下のネットワークが指定されている．

- 3 つの隠れ層を持つ．
- 各層のノード数は，入力層から出力層まで順番に 14，20，12，12，2 である．
- 学習時のミニバッチサイズは 100 である．

また，前処理として，各メトリクスを 0–1 範囲にスケーリングする．計算式は以下である．

$$\mathbf{X}_{0-1} = \frac{\mathbf{X}_{org} - X_{\min}}{X_{\max} - X_{\min}} \quad (1)$$

各メトリクスに対して，$\mathbf{X}_{org}$ は全変更に対するそのメトリクス値のベクトル，$X_{\min}$ は全変更の中で最小のメトリクス値，$X_{\max}$ は最大のメトリクス値である．

学習の繰返し回数は，Deeper の説明に記載されていない．そのため，誤差をプロットし繰返し回数を決定する．結果，50 回の繰返しにより，Deeper の汎化を十分行えることが分かったため，Deeper の繰返し回数を 50 とする．

4.4 データのリサンプリング

表 2 に示すとおり，不具合を混入した変更は全体の変更の数に対して少ない傾向にある．クラスごとのデータ数の不均一は多数派のクラスにバイアスをかけ，予測モデルの学習精度を下げてしまう可能性がある．不具合予測におい

© 2018 Information Processing Society of Japan

表 3 リサンプリング済みのプロジェクトの変更数
Table 3 The number of changes of resampled datasets.

Project	Buggy Changes	Clean Changes
Hadoop	3,280	3,280
Camel	5,620	5,620
Gerrit	3,470	3,470
Osmand	4,150	4,150
CMake	2,790	2,790
Bitcoin	1,450	1,450
Gimp	8,080	8,080

ては，データのリサンプリングにより精度を上げることが可能であると知られており [22]，その評価においてもデータ数を均一にするためのリサンプリング技術が用いられている [9], [22], [25].

本研究では各クラスのデータ数を統一する手法として，ランダムアンダーサンプリングを用いる．ランダムアンダーサンプリングは，データ数が各クラスで均一になるまで，データ数の多いクラスからランダムにデータを削除していく手法である．**表 3** にリサンプリング後のプロジェクトごとの変更数を示す．ここで，変更数が表 2 から計算できる変更数と異なることが分かる．たとえば，Hadoop であれば，不具合を含んだ変更が 24.8%あり，各クラスのデータが 3,452 個に揃わなくてはならない．これは，すべての変更にソースコードの変更が含まれているわけではないためである．そのため，各プロジェクトで，表 2 から計算できる変更数よりも少ない変更数となっている．また，すべてのプロジェクトで 1 桁目の端数は丸めている．

4.5 10 × 10 重交差検証

本研究では実験におけるデータの選択の偏りを減らすために，訓練データ，および，テストデータの選択に 10 × 10 重交差検証を利用する．10 × 10 重交差検証は，10 重交差検証を 10 回実行する評価手法であり，不具合予測手法の研究で多く利用されている [25].

それぞれの回でランダムアンダーサンプリングを再度実行することでデータの偏りをできる限り減らすことを行う．最後に，10 × 10 重交差検証の結果の平均値を計算し，各々の手法の評価値とする．評価値には閾値に依存しない ROC（Receiver Operating Characteristic）曲線から得られる AUC（Area Under the Curve）を利用する．

5. 結果

5.1 RQ1：W-CNN による不具合の学習・予測は可能か？

5.1.1 動機

我々の知る限り，今までに変更のソースコード片に対して深層学習を適用した研究は行われていない．そのため，まず，提案手法 W-CNN が変更のソースコード片に対する不具合予測に適用可能かどうかを調べる．

また，学習が可能であるならば，どの程度の W-CNN の学習の繰返し回数（本研究ではエポック数を単位としている）が適切であるのかを調査する．エポック数とは，W-CNN における学習を何回繰り返すかの値である．通常，エポック数が大きければ大きいほど，より訓練データに適合した CNN を生成できる．しかし，エポック数が大きすぎると，ネットワークが訓練データに特化しすぎてしまい，テストデータに対して精度が悪くなる（汎化性能が悪くなる）という過適合の問題が発生する．したがって，汎化性能を大きくし，かつ，過適合を起こさない最大のエポック数を求める．

5.1.2 アプローチ

W-CNN が不具合を学習可能か，また過適合を起こさずに汎化できているかを調査するため，学習精度を示す訓練誤差（Train loss），および，汎化性能を示すテスト誤差（Test loss）を計算する．この計算のために，我々の手法では交差エントロピーを用いている．50 エポックまで学習を進めて，訓練誤差，および，テスト誤差をプロットし，その学習精度，および，汎化性能を調べる．エポック数が増えているにもかかわらず，テスト誤差が一定か大きくなる場合は過適合の恐れがあると判断する．

さらに，AUC の値を 5 エポックごとに記録することで，分類精度の推移を調べる．これにより，適切なエポック数を選択する．AUC の値は 10 × 10 重交差検証によりプロジェクトごとに 100 回実行が行われた結果の平均とする．

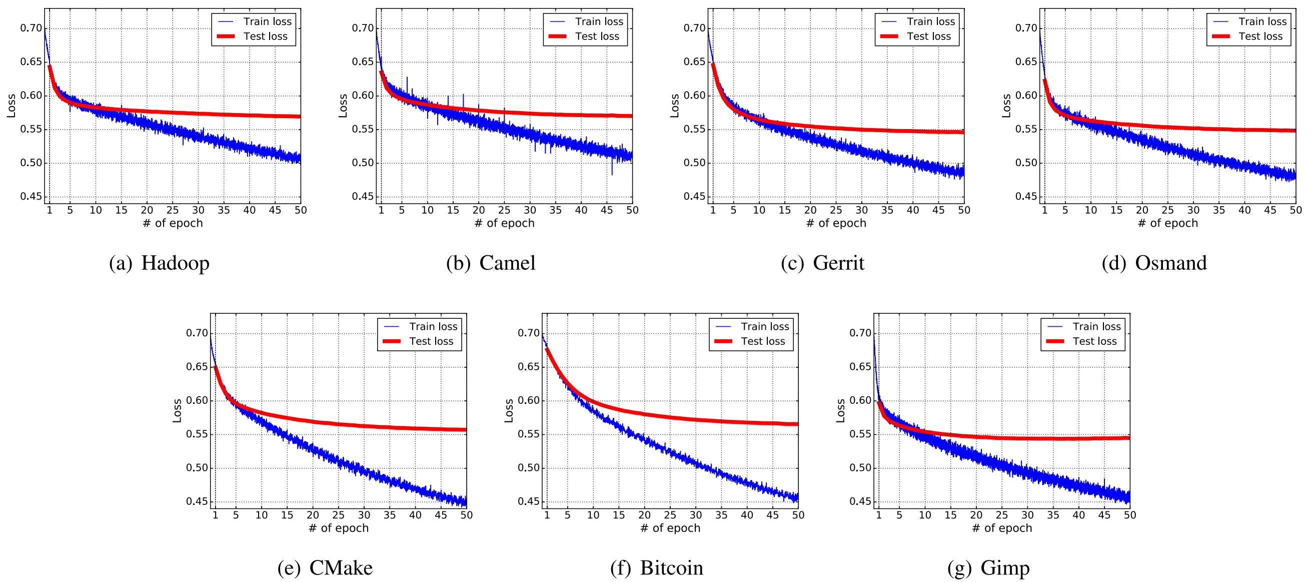
5.1.3 結果

学習が進むごとに訓練誤差，および，テスト誤差が減少していることから，提案手法 W-CNN は変更のソースコード片に対する不具合予測に適用可能である．

図 5 に，各プロジェクトの学習過程における訓練誤差，および，テスト誤差を示す．この図から，学習を進めるごとにすべてのプロジェクトにおいて訓練誤差，および，テスト誤差が減少していることが分かる．訓練誤差が小さければ学習が進んでいることを示し，テスト誤差が小さければテストデータに適合（汎化）していることを示している．

ただし，15 エポック以降は学習データに対する学習は進んでいるが，汎化は進みにくくなることが確認できる．これは，図 5 より 4 つのプロジェクト（Hadoop，Camel，Gerrit，Osmand）でエポック数が 15 を超える程度で訓練誤差，および，テスト誤差の差が大きくなり始めていることから確認できる．また，残りの 3 つのプロジェクト（CMake，Bitcoin，Gimp）ではより早い段階でこの傾向が見られる．これらより，エポック数 15 を学習が進んだ W-CNN として以降の RQ で利用する 1 つ目のエポック数とする．

© 2018 Information Processing Society of Japan



(a) Hadoop　(b) Camel　(c) Gerrit　(d) Osmand

(e) CMake　(f) Bitcoin　(g) Gimp

図 5　各プロジェクトの学習過程における W-CNN の訓練誤差（Train loss）およびテスト誤差（Test loss）

Fig. 5　Train loss and test loss for each project in W-CNN learning process.

表 4　学習過程における W-CNN の AUC のプロジェクト間での平均値

Table 4　Average AUC values in W-CNN learning process.

エポック数	5	10	15	20	25	30	35	40	45	50
AUC	0.788	0.797	0.803	0.807	0.811	0.813	0.815	0.816	0.816	0.817

> **学習が進むごとに AUC の値が上昇していることが確認でき，最高で平均 AUC 0.817 が得られる．**

表 4 は各プロジェクトで得られた AUC の値の平均を 5 エポックごとにまとめた表である．学習が進むごとに AUC の値が上昇していることが分かる．一方で，進むごとに上昇幅は小さくなることも確認できる．エポック数 40 から 50 の範囲ではほとんど変化が見られない．そこで，エポック数 50 を学習を十分進めた W-CNN として以降の RQ で利用する 2 つ目のエポック数とする．

提案手法 W-CNN の学習過程を記録した結果，W-CNN は本実験で用意した学習データに対して適用可能であり，平均 AUC は最高で 0.817 であることから，提案手法による不具合分類が可能であることが分かる．また，得られた結果からエポック数 15 と 50 の時点のモデルを以降の RQ で用いることとする．

5.2　RQ2：W-CNN は既存の変更に対する深層学習を用いた不具合予測手法よりも予測精度が良いか？

5.2.1　動機

提案手法 W-CNN が不具合予測として既存の手法よりも精度が良いか否かを調べる．これにより，W-CNN による不具合予測が効果的であるかを調べる．

ここで，我々が立てた仮説は「大規模なデータを利用した CNN によって，バグ検出の精度が向上するのではないか」というものである．先行手法として本研究でも比較対象とする Deeper においては 14 個の変更メトリクスから Deep Belief Network を構築しており，入力としては比較的小さいものとなっている．大規模な特徴抽出をテキストから行うことで，バグ検出精度が向上することを期待する．

5.2.2　アプローチ

比較対象として，従来の不具合予測手法で我々の手法とアプローチが最も近い Yang らの Deeper [25] を用いる．Deeper も，変更に対して深層学習を用い不具合予測を行うが，変更メトリクスを用いた点や，深層学習のモデルなどが異なる．提案手法に関しては，RQ1 で示した 2 つのエポック数（15，および，50）における W-CNN（以降，それぞれ W-CNN 15，および，W-CNN 50 と呼ぶ）を用いる．評価値には AUC を用いる．この値は 10×10 重交差検証を行った平均値として計算する．

5.2.3　結果

> **W-CNN 15，および，W-CNN 50 は，既存の深層学習による変更に対する不具合予測手法よりも予測精度が良い．**

表 5 は，W-CNN 15，W-CNN 50，および，Deeper から得られた AUC の値をまとめた表である．すべてのプロジェクトにおいて，W-CNN 50 が最良であることが分か

© 2018 Information Processing Society of Japan

表 5　W-CNN 15，W-CNN 50，および，Deeper の AUC
Table 5　AUC values by W-CNN 15, W-CNN 50, and Deeper.

	提案手法		従来手法
プロジェクト	W-CNN 15	W-CNN 50	Deeper
Hadoop	0.788	0.802	0.684
Camel	0.783	0.798	0.630
Gerrit	0.817	0.832	0.752
Osmand	0.810	0.825	0.698
CMake	0.803	0.821	0.622
Bitcoin	0.793	0.811	0.675
Gimp	0.825	0.830	0.624
AVG	0.803	0.817	0.669

る．また，W-CNN 15 も Deeper より予測精度が高いことが確認できる．以上より，W-CNN は既存の手法より高い精度をソースコード片のみから学習可能であることが分かる．この結果から，多くの入力データを利用することで予測精度が改善できる可能性が示された．

以上の実験の過程で，Deeper と W-CNN に関する興味深い差異が見つかった．Deeper では変更行数が多いものを不具合あり，変更行数が少ないものを不具合なしと判定する傾向が強かった．一方，W-CNN では変更行数の違いによる判定傾向に差は出なかった．これは W-CNN は変更行数の影響を受けにくいことを示している．より詳細な分析にはさらなる実験が必要であるため，今後の課題としたい．

5.3　RQ3：W-CNN の学習・予測時間はどの程度か？

5.3.1　動機

一般に深層学習はロジスティック回帰などの軽量な機械学習よりも学習の時間を必要とする．また，巨大なネットワークを持つ深層学習は，小規模なネットワークを持つ深層学習よりも学習時間がかかる．我々は，W-CNN と Yang らの Deeper [25] の学習および予測時間を比較して，提案手法を考察する．

5.3.2　アプローチ

学習が終了するまでの時間を学習時間として 1 重交差検証ごとに計測する．つまり，W-CNN 15 ならば，15 回の学習のエポックが終了するまでの時間である．この時間を，10×10 重交差検証を行い 100 回計測し，その平均値を評価する．実行環境は，Intel Xeon CPU E5-1620 v3 @ 3.50 GHz，NVIDIA GeForce GTX TITAN X（3584 cuda cores, 12 GB）である．

5.3.3　結果

W-CNN 15，および，W-CNN 50 は既存手法よりも長い学習時間を必要とするが，各コミットに対する予測時間は短い．

表 6　W-CNN 15，W-CNN 50，および，Deeper の学習時間（秒）
Table 6　Training times by W-CNN 15, W-CNN 50, and Deeper (s).

	提案手法		従来手法
プロジェクト	W-CNN 15	W-CNN 50	Deeper
Hadoop	260.5	625.4	54.6
Camel	363.9	1,006.2	92.7
Gerrit	300.9	692.8	71.4
Osmand	350.5	830.0	118.1
CMake	263.0	588.7	107.7
Bitcoin	219.0	384.1	45.5
Gimp	502.6	1,439.6	138.3
AVG	322.9	795.3	89.8

表 7　学習後の W-CNN，および，Deeper の 1 コミットに対する識別時間（秒）
Table 7　Prediction times by W-CNN and Deeper for one commit (s).

	提案手法	従来手法
プロジェクト	W-CNN	Deeper
Hadoop	0.0004	0.0008
Camel	0.0003	0.0008
Gerrit	0.0004	0.0008
Osmand	0.0004	0.0008
CMake	0.0004	0.0008
Bitcoin	0.0004	0.0009
Gimp	0.0004	0.0008
AVG	0.0004	0.0008

表 6 は，W-CNN 15，W-CNN 50，および，Deeper から得られた 1 フォールドの学習時間をまとめた表である．W-CNN 15 は Deeper よりも，また，W-CNN 50 は W-CNN 15 よりも，長い学習時間がかかる．そのため，学習時間の点では，Deeper が最も良い．ただし，最も学習に時間がかかる Gimp であっても，学習時間は 23 分程度である．また，**表 7** に示しているとおり，一度学習すれば，その後の各コミットに対する予測時間は，提案手法で 0.0004 秒程度などと短い．そのため，変更に対する実行時間のオーバヘッドは少ない．

W-CNN の学習時間は，変更数（コミット数）によらないが，既存手法である Deeper は変更数と強く関係がある．

表 3 と合わせて考えると，従来手法である Deeper は変更数が多いプロジェクトほど多くの学習時間がかかる傾向があるが，W-CNN はその傾向が小さい．これは，W-CNN の分析対象が変更のソースコード片であり，その長さが学習時間に影響を与えているためである．

© 2018 Information Processing Society of Japan

6. 妥当性の検証

6.1 構成概念妥当性

評価指標として AUC を用いている．他に不具合予測で一般的に利用される評価指標として Precision や Recall，これらの組合せである F1 値があるが，AUC はこれらの指標の要素を表現可能な指標であり，不具合予測の評価指標として妥当であると考える．

また，本実験では，データの選択の偏りを取り除くために 10×10 重交差検証を行っている．そのため，データの選択の偏りも小さくできていると考える．

本実験では，考慮する文脈行数を 3 行として実験を行っている．これは，Hadoop プロジェクトに対して文脈行数が 3 行のときに最も良い識別結果であったこと，および，Git の diff コマンドで標準で出力される行数が 3 行であることによる．一方で，他の文脈行数が最も良い識別性能を出す場合もあり，文脈行数 3 行が最も良い文脈行数であるとは断定できない．しかし，本研究の目的は，変更のソースコード片に対して深層学習を適用し不具合予測を行うことであり，どの文脈行が最も優れているかを検証することではない．そのため，最適な文脈行に関する検証は今後の研究課題として考えている．

6.2 外的妥当性

本実験では 7 つのソフトウェアプロジェクトを選択し，それらから収集した変更を実験データとして利用している．選択したプロジェクトは，C，および，Java の 2 つの言語のどちらかで書かれており，また，様々な分野（サーバ，Web アプリケーション，モバイルアプリケーション，開発ツール，デスクトップアプリケーション）をカバーしている．しかし，これらのプロジェクトだけでは，世にあるすべてのソフトウェアプロジェクトに対しての一般的な結果を示せていない可能性がある．より対象プロジェクトを増やすことで，外的妥当性を高めることができる．また，本実験の対象プロジェクトはすべて OSS であるため，実開発組織と共同研究による妥当性の向上が必要である．

6.3 内的妥当性

本研究の実験ではデータへのラベル付けに Commit Guru を利用している．しかし，Commit Guru による不具合混入コミットの網羅率は完全ではない．たとえば，不具合を修正したコミットのコミットメッセージに特定のキーワードが含まれていなかった場合は，その不具合を見逃してしまう．一方で，公開されている Web アプリケーションであり，実験の再現性が高く，また今後の応用も行いやすいと考える．

実験に利用したスクリプトに対して検証は行っているが，我々が気づいていない不具合が含まれている可能性はある．特に，比較評価に用いた Yang らの Deeper はスクリプトなどが公開されておらず，実装が完全に再現できているかを確認する手段がない．

7. 結論

本論文では，深層学習の分類モデルである畳み込みニューラルネットワークを用いた不具合予測手法 W-CNN を提案した．我々の知る限り，提案手法は，変更のソースコード片に対して深層学習を適用し不具合予測を行った初めての研究である．また，提案手法の評価のために 7 つのソフトウェアプロジェクトに対して，提案手法と従来の変更に対する深層学習を用いた不具合予測手法 Deeper の比較実験を行った．実験から得られた本研究の貢献を以下にまとめる．

(1) 提案手法 W-CNN の学習過程の訓練誤差，および，テスト誤差を調べることにより，変更のソースコード片のみを用いた深層学習によって不具合の予測が可能であり，さらに，その分類精度が高いことを示した．
(2) 提案手法 W-CNN は，既存の変更に対する深層学習を用いた不具合予測手法と比較して，学習に時間がかかるが，コミットごとの予測時間は短く，高い不具合予測精度をもたらすことが可能であることを確認した．

以上の結果から，W-CNN は分類精度の高い不具合予測手法として期待できる．また，W-CNN は学習に時間がかかるが，1 度学習を終えると以降の各コミットに対しての不具合である確率計算は 0.0004 秒程度と非常に短い時間で実行することが可能であり，実用に足る効率的な不具合予測手法になりうると考える．

参考文献

[1] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., et al.: Tensorflow: Large-scale machine learning on heterogeneous distributed systems, arXiv:1603.04467 (2016).

[2] Aversano, L., Cerulo, L. and Del Grosso, C.: Learning from bug-introducing changes to prevent fault prone code, *IWPSE*, pp.19–26, ACM (2007).

[3] Brown, P.F., Desouza, P.V., Mercer, R.L., Pietra, V.J.D. and Lai, J.C.: Class-based n-gram models of natural language, *Computational Linguistics*, Vol.18, No.4, pp.467–479 (1992).

[4] Dos Santos, C.N. and Gatti, M.: Deep Convolutional Neural Networks for Sentiment Analysis of Short Texts, *COLING*, pp.69–78 (2014).

[5] Hindle, A., German, D.M. and Holt, R.: What do large commits tell us?: A taxonomical study of large commits, *MSR*, pp.99–108, ACM (2008).

[6] Ihaka, R. and Gentleman, R.: R: A language for data analysis and graphics, *Journal of Computational and Graphical Statistics*, Vol.5, No.3, pp.299–314 (1996).

[7] Jiang, T., Tan, L. and Kim, S.: Personalized defect prediction, *ASE*, pp.279–289, IEEE (2013).

[8] Kamei, Y. and Shihab, E.: Defect prediction: Accom-

© 2018 Information Processing Society of Japan

plishments and future challenges, *SANER*, Vol.5, pp.33–45, IEEE (2016).
[9] Kamei, Y., Shihab, E., Adams, B., Hassan, A.E., Mockus, A., Sinha, A. and Ubayashi, N.: A Large-Scale Empirical Study of Just-in-Time Quality Assurance, *IEEE Trans. Software Engineering*, Vol.39, No.6, pp.757–773 (2013).
[10] Kim, S., Whitehead, Jr., E.J. and Zhang, Y.: Classifying software changes: Clean or buggy?, *IEEE Trans. Software Engineering*, Vol.34, No.2, pp.181–196 (2008).
[11] Kim, Y.: Convolutional neural networks for sentence classification, arXiv:1408.5882 (2014).
[12] Kingma, D. and Ba, J.: Adam: A method for stochastic optimization, arXiv:1412.6980 (2014).
[13] Krizhevsky, A., Sutskever, I. and Hinton, G.E.: Imagenet classification with deep convolutional neural networks, *NIPS*, pp.1097–1105 (2012).
[14] Lam, A.N., Nguyen, A.T., Nguyen, H.A. and Nguyen, T.N.: Combining deep learning with information retrieval to localize buggy files for bug reports (n), *ASE*, pp.476–481, IEEE (2015).
[15] Menzies, T., Krishna, R. and Pryor, D.: The Promise Repository of Empirical Software Engineering Data, available from ⟨http://openscience.us/repo⟩ (accessed 2017-10-17).
[16] Mikolov, T.: word2vec: Tool for computing continuous distributed representations of words, available from ⟨https://code.google.com/archive/p/word2vec/⟩ (accessed 2017-10-17).
[17] Mizuno, O. and Kikuno, T.: Training on errors experiment to detect fault-prone software modules by spam filter, *ESEC/FSE*, pp.405–414, ACM (2007).
[18] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al.: Scikit-learn: Machine learning in Python, *Journal of Machine Learning Research*, Vol.12, No.10, pp.2825–2830 (2011).
[19] Pennington, J., Socher, R. and Manning, C.D.: Glove: Global Vectors for Word Representation, *EMNLP*, Vol.14, pp.1532–1543 (2014).
[20] Rosen, C., Grawi, B. and Shihab, E.: Commit Guru: Analytics and Risk Prediction of Software Commits, *ESEC/FSE*, pp.966–969, ACM (2015).
[21] Śliwerski, J., Zimmermann, T. and Zeller, A.: When do changes induce fixes?, *Sigsoft Software Engineering Notes*, Vol.30, No.4, pp.1–5, ACM (2005).
[22] Tan, M., Tan, L., Dara, S. and Mayeux, C.: Online defect prediction for imbalanced data, *ICSE*, pp.99–108, IEEE (2015).
[23] Thomas, W.S.: lscp: A lightweight source code preprocesser, available from ⟨https://github.com/doofuslarge/lscp⟩ (accessed 2017-10-17).
[24] Wang, S., Liu, T. and Tan, L.: Automatically learning semantic features for defect prediction, *ICSE*, pp.297–308, ACM (2016).
[25] Yang, X., Lo, D., Xia, X., Zhang, Y. and Sun, J.: Deep learning for just-in-time defect prediction, *QRS*, pp.17–26, IEEE (2015).
[26] Zhang, X., Zhao, J. and LeCun, Y.: Character-level convolutional networks for text classification, *NIPS*, pp.649–657 (2015).
[27] Zimmermann, T., Premraj, R. and Zeller, A.: Predicting defects for eclipse, *PROMISE*, p.9, IEEE (2007).

近藤 将成

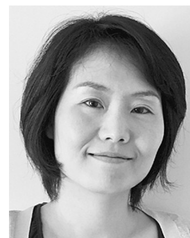
平成 29 年京都工芸繊維大学工芸科学部情報工学課程卒業．学士（工学）．同大学大学院工芸科学研究科情報工学専攻博士前期課程在学中．平成 29 年国立研究開発法人産業技術総合研究所リサーチアシスタント．教師なし学習モデルによるソフトウェア不具合予測に関する研究に従事．

森 啓太

平成 29 年京都工芸繊維大学大学院工芸科学研究科情報工学専攻博士前期課程修了．修士（工学）．CNN のソフトウェア工学への応用に関する研究に従事．同年 4 月より三菱電機（株）．

水野 修 （正会員）

平成 11 年大阪大学大学院基礎工学研究科助手．平成 13 年博士（工学）大阪大学．平成 22 年京都工芸繊維大学大学院工芸科学研究科准教授．平成 29 年同大学情報工学・人間科学系教授．主にソフトウェアのバグ検出手法，ソフトウェアリポジトリのマイニングに関する研究に従事．



崔 銀惠 （正会員）

平成 14 年大阪大学大学院情報数理系専攻博士後期課程修了．博士（工学）大阪大学．同年（株）東芝研究開発センター．平成 16 年より産業技術総合研究所特別研究員，研究員を経て，現在，主任研究員．主にソフトウェアのテスト，検証，不具合特定に関する研究に従事．

© 2018 Information Processing Society of Japan