修 士 論 文

題　目　　深層学習による不具合混入コミットの予測モデルの提案とその評価

主任指導教員　　水野　修　教授

京都工芸繊維大学大学院 工芸科学研究科

情報工学専攻

学生番号　　15622051

氏　　名　　森　啓太

平成29年2月10日提出

様式 2 号

学位論文の要旨（和文）

平成 29 年 2 月 10 日

京都工芸繊維大学大学院
工芸科学研究科長　殿

工芸科学研究科　　情報工学専攻
平成 27 年入学
学生番号　　15622051
氏　　名　　森　啓太　㊞

（主任指導教員　　水野　修　㊞　）

本学学位規則第 4 条に基づき、下記のとおり学位論文内容の要旨を提出いたします。

1.　論文題目

深層学習による不具合混入コミットの予測モデルの提案とその評価

2.　論文内容の要旨（400 字程度）

ソフトウェアの品質保証活動の向上のため不具合予測は注目されている分野であり，これまで不具合予測モデルに関して多くの研究が行われてきた．また最近の研究では，より良い性能を得るために不具合予測モデルにも深層学習を適用しようとする試みがなされ始めてきている．しかしそれらの研究では深層学習を特徴の生成モデルとしてしか利用しておらず，我々が知る限りでは，深層学習を分類器として不具合予測モデルに適用した研究はまだ行われていない．そこで本研究では深層学習による分類器の不具合予測への適用可能性と予測性能を調査するため，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 (CNN) を用いた不具合予測モデル W-CNN を提案した．モデルの評価実験の結果，提案手法 W-CNN は不具合の学習と予測が可能であり，深層学習による分類器は不具合予測に適用可能であることがわかった．また，W-CNN は従来の不具合予測モデルにも劣らない性能を持っていることもわかった．

An Application of Deep Learning Based Classifier to Defect Prediction

2017 15622051 *MORI Keita*

Abstract

Much research on defect prediction model has been done so far to improve software quality assurance activities. Although newly proposed models can predict defects well, researchers still search better defect prediction models. Recently, some research tries to leverage Deep Learning technique in defect prediction area. While these research use generative model of Deep Learning technology (e.g. Deep Belief Network) to extract better features, as far as we know, none of them builds a prediction model using classification model (e.g. Convolutional Neural Network (CNN)). This paper proposes W-CNN, a change-level defect prediction model using CNN. We perform defect predictions on seven open source software projects. Then, we evaluate prediction performance of W-CNN and six baseline models to investigate whether classification model of Deep Learning can be applied to defect prediction. The results show that W-CNN can learn defect and classify buggy changes, and CNN can be applicable for defect prediction. Additionally, we find W-CNN stands comparison with conventional defect prediction models.

目　次

1. 緒言 **1**

2. 関連研究と背景 **4**

2.1 関連研究 4

2.1.1 不具合予測モデル 4

2.1.2 ソフトウェア工学への深層学習の適用事例 4

2.2 提案手法 W-CNN: 畳み込みニューラルネットワーク (CNN) によるテキスト分類器を利用した不具合予測モデル 5

2.2.1 前処理 7

2.2.2 ネットワークの構造 7

2.2.3 ハイパーパラメータと訓練アルゴリズム 8

2.3 Commit Guru 8

2.3.1 不具合混入コミットのラベル付け 9

2.3.2 メトリクスの収集 9

3. 研究の方法 **10**

3.1 対象とするプロジェクトデータ 10

3.2 ベースラインモデル 12

3.2.1 従来の変更レベルの不具合予測モデル 12

3.2.2 従来のテキスト分類モデル 12

3.3 データの準備 14

3.4 実験データセットの作成 14

3.4.1 ソースコードを変更していない変更の削除 14

3.4.2 データのリサンプリング 16

3.5 モデルの性能評価 16

3.5.1 10 タイムズ 10 フォールドクロスバリデーション 16

3.5.2 パラメータのチューニング 16

3.5.3 評価指標の計算 18

4. 結果 **21**

4.1 RQ1: 提案手法 W-CNN による不具合の学習は可能か？ 21

4.1.1 概要 21

4.1.2 結果 21

4.2 RQ2: 提案手法 W-CNN は既存の不具合予測モデルより性能が良いか？ 22

4.2.1 概要 22

4.2.2 結果 25

4.3 RQ3: 提案手法 W-CNN は既存のテキスト分類器よりもソースコード片の分類能力が高いか？ 28

4.3.1 概要 28

4.3.2 結果 28

4.4 RQ4: 提案手法 W-CNN のタイムコストはどの程度か？ 29

4.4.1 概要 29

4.4.2 結果 29

5. 妥当性の検証 **34**

5.1 構成概念妥当性 34

5.2 外的妥当性 34

5.3 内的妥当性 34

6. 結言 **36**

謝辞 **37**

参考文献 **38**

1. 緒言

ソフトウェアは我々の生活に大きな影響を与えており，それらの品質を保つためのテストやレビューといった品質保証活動は重要である．そういった品質保証を効率的に行うため，不具合予測モデルの研究がなされてきた [1–10]．不具合予測モデルは，過去のソフトウェア開発の履歴情報等を用いることで，不具合を含んでいる可能性が高い場所を予測するものである．その予測結果によって不具合を早期に発見することやメンテナンス工数の割当の優先付けをすることができる．近年の機械学習ツール [11–13] の発展，PROMISE データセット [14] やリポジトリを解析しデータを提供する不具合予測ツール Commit Guru [15,16] の公開などもあり，不具合予測モデルに関する研究はより進んできている分野である [17].

予測モデルの構築において，性能に大きな影響を与える学習アルゴリズムの選択は重要な点である．中でも最近では不具合予測に深層学習を適用した研究 [7,8] がある．これらの研究は深層学習アルゴリズムの一つで生成モデル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より有効な特徴セットを生成できることを示した．深層学習に関しては近年多くの研究 [18–21] がなされており，特に分類問題に関して成功を収めている．不具合予測モデルにおいてもより良い性能を得るため，さらに深層学習を適用していく必要があるはずである．

前述の通りこれまでの深層学習を不具合予測へ適用した研究では，主に深層学習を用いてより良い特徴を生成することに重きがおかれていた．そのため現時点で，深層学習は分類器として利用されていない．そこで本研究では，深層学習を特徴生成器ではなく不具合分類器そのものとして適用し，深層学習による分類器の不具合予測モデルへの適用可能性と不具合予測性能を調査する．深層学習アルゴリズムの中でも，分類器としては畳み込みニューラルネットワーク (CNN) が特に成果を上げている．畳み込みニューラルネットワークは主に画像のカテゴリ分類での成功 [18] が大きいが，近年ではテキスト分類でも成果を出している [19–21]．また不具合予測モデルにおいては，テキスト分類を用いたモデルがすでに多数提案されており [2–5,8]，ソースコード等のテキスト情報は不具合予測に利用可能であることがわかっている．そこで我々は深層学習による分類器を不具合予測に適用するにあたり，テキスト分

類器の適用の容易さを考慮し，Kim 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 [20] に注目した．このモデルはテキストを単語レベルの情報を用いて畳み込みニューラルネットワークで分類するもので，特に 2 極性分類において，比較対象であった他のモデルよりも高い分類性能を記録していた．また，これまでの変更 (コミット) レベルの不具合予測モデル [2,4–7] (変更が不具合を混入したかどうかを予測するモデル) の利点を考慮し，我々も変更レベルの不具合予測モデルを構築する．変更レベルの不具合予測モデルは，変更が行われた段階でその変更のリスクを予測することができ，開発者に対して早くフィードバックを与えることが出来る．さらに開発者は不具合であると予測された変更に関するソースコードのみをレビューするだけで良く，レビューが簡単であるという利点もある．我々は Kim の分類モデル [20] を基に，変更により修正・追加されたソースコードを入力とする変更レベルの不具合予測モデル W-CNN(Word CNN) を提案する．

本研究では，W-CNN を評価するため，以下の研究設問を設定した．

- **RQ1:** 提案手法 **W-CNN** による不具合の学習は可能か？
- **RQ2:** 提案手法 **W-CNN** は既存の不具合予測モデルより性能が良いか？
- **RQ3:** 提案手法 **W-CNN** は既存のテキスト分類器よりもソースコード片の分類能力が高いか？
- **RQ4:** 提案手法 **W-CNN** のタイムコストはどの程度か？

これらの研究設問に答えるため，我々は 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意し，W-CNN と比較対象である 6 つのベースラインモデルに対して不具合予測性能の評価実験を行った．その結果 RQ1 では，提案手法 W-CNN の学習過程を記録した結果，W-CNN は学習データに対し適合可能であり，不具合の分類も可能であることがわかった．RQ2 では，W-CNN は 2 つの既存の不具合予測モデルと比較して最も良い性能を持つモデルであることがわかった．また RQ3 では，W-CNN は 4 つの既存のテキスト分類モデルと比較しても最も良い性能を持つモデルであることがわかった．RQ4 では，W-CNN は他の 6 つのベースラインモデルよりも学習にタイムコストがかかってしまうことがわかった．

本稿の主な貢献を以下に纏める．

1. 我々が知る限り，深層学習を分類器として初めて不具合予測に適用した．

2. 深層学習による分類器は不具合予測に適用可能であり，タイムコストはかかるが既存の不具合予測モデルよりも高い性能を出し得る能力があることを確認した．この結果から，深層学習による分類器の適用は不具合予測モデルを向上させるアプローチとして期待できるものであることを示した．
3. Web アプリケーションとして公開されている Commit Guru を利用することにより，容易に不具合のラベル付けや比較実験を行えることを示した．

以降の本稿の構成を紹介する．第 2 章では本研究に関連する研究と本研究の背景となる深層学習モデルや Commit Guru について述べる．第 3 章では本研究の実験手法について述べる．第 4 章ではそれぞれの研究設問に対する結果をまとめる．第 5 章では本研究の妥当性の検証を行う．第 6 章では本研究の結論を述べる．

2. 関連研究と背景

2.1 関連研究

2.1.1 不具合予測モデル

不具合予測はソフトウェアのメンテナンスの効率化を実現するために重要な分野であり，これまで様々な不具合予測モデルが提案されてきた [1–8]．多くの不具合予測モデルは機械学習を用いて構築され，コード行数，複雑度，修正の頻度などのような様々なソフトウェアメトリクスが特徴として使われてきた．Zimmermann らの研究 [1] では，ソースコードに関する複雑度の組み合わせを用いて予測モデルを構築しており，Kamei らの研究 [6] では，変更に関するメトリクスの組み合わせを用いて予測モデルを構築している．こうした中で，ソースコード等のテキストに注目した不具合予測モデルも研究されている [2–5,8]．テキストは複雑度メトリクス等に比べ収集が容易であり，テキスト分類技術も日々進歩していることから，不具合予測にテキスト分類技術が用いられることは多く見られる．Mizuno らの研究 [3] 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モデルを構築した．また，我々の提案手法に似たモデルとして，変更に関するテキスト情報を用いる不具合予測モデルの研究がある [2,4,5]．これらは変更されたソースコードやファイルのパスを学習することによって，変更が不具合を混入したかを予測する手法であり，機械学習テクニックとして k 近傍法やサポートベクトルマシン等を用いている．本研究では，深層学習によるテキスト分類器の性能に期待して，Kim 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 [20] を用いた不具合予測モデルを構築する．

2.1.2 ソフトウェア工学への深層学習の適用事例

近年ソフトウェア工学においても，深層学習が適用され始めている．Yang らの研究 [7] は，生成モデルであるディープビリーフネットワークを用いることによって，Kamei らの不具合予測モデル [6] で利用されていたメトリクスから，モデルの性能を向上させるより有効な特徴セットを生成できたことを示した．また，深層学習モデルにソースコードやバグレポート等のソフトウェアにおけるテキスト情報を学習

させている研究もある．Wang らの研究 [8] は，ソースコードの特徴生成にディープビリーフネットワークを用いた場合，ソースコードの意味的な違いをより表現でき，不具合予測の性能を向上させることができることを示した．Lam らの研究 [22] は，バグレポートからの不具合箇所の推定を行うために，深層学習と IR 技術を結合した類似性計測モデルを提案し，深層学習を使うことで既存の IR 技術より高い性能を得ることができたことを示した．これらの研究は深層学習を用いた特徴生成や特徴の結合に重きを置いている．我々は畳み込みニューラルネットワークのみからなる，深層学習モデルを分類器として用いたシンプルなテキストベースの不具合予測モデルを提案する．

2.2 提案手法 W-CNN: 畳み込みニューラルネットワーク (CNN) によるテキスト分類器を利用した不具合予測モデル

深層学習アルゴリズムの一種である畳み込みニューラルネットワーク (CNN) は深層学習の分類モデルとして画像分類で成功を収めてきたアルゴリズムであり [18]，最近ではテキスト分類においても高い精度を得ている [19–21]．テキスト分類技術は不具合予測に有効であることがわかっており [2–5,8]，不具合予測への適用も容易であることから，我々は深層学習のテキスト分類モデルに注目した．一般的に深層学習は層の数が多い程大量の学習データと時間を要する傾向にあり，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 [21] では，隠れ層が 9 層のネットワークで訓練データには 100 万個程度のデータを用いている．対して，我々が対象とする変更レベルの不具合予測で利用できる学習データ数は比較的大きなプロジェクトを利用しても数千から 1 万個程度である．そのため，我々は Kim により提案されたテキスト分類モデル [20] を利用することとした．このモデルは隠れ層が 3 層のネットワークで構成されていることから，その他のより深いネットワークで構成されるモデルよりも軽量である．Kim は 4 千から 1 万個程度の訓練データでネットワークを十分に学習させていたことから，本研究で用いるデータにも適用可能と判断した．我々は Kim のモデル [20] を基に，ソースコードの単語レベルの情報を用いて分類器を構成する不具合予測モデル W-CNN(Word CNN) を作成した．図 2.1 に提案手法 W-CNN で行われる処理の概要を示す．W-CNN の実装は TensorFlow [13] を

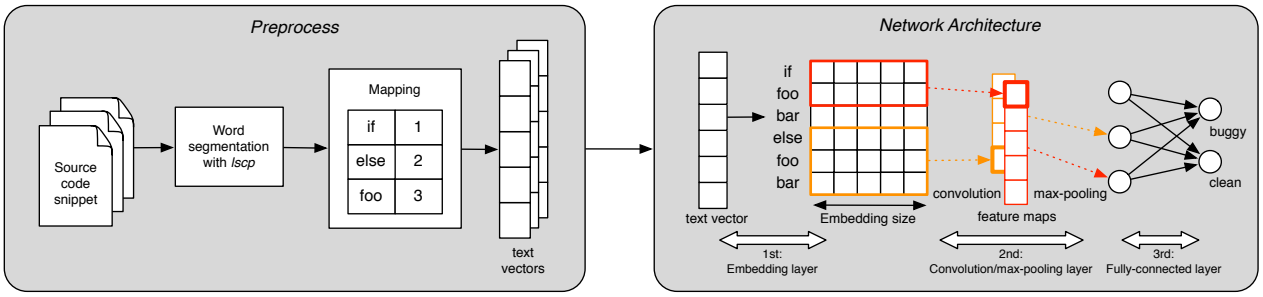


図 2.1: 提案モデル W-CNN の処理の概要

用いて行った．

2.2.1 前処理

本モデルの入力は変更により修正・追加されたソースコードをつなぎ合わせたソースコード片である．我々はソースコードを単語分割するために Thomas が公開している lscp(A lightweight source code preprocesser) [23] を用いた．lscp はソースコードを構文解析せず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．前処理では，まず訓練データのソースコード片から単語を収集する．次にそれらの単語に対して，頻出ワード順に 1 から数字をマッピングし，マッピングテーブルを作成する．その後訓練データ，テストデータ共にマッピングテーブルを使用してテキストベクトルへの変換を行う．ただしマッピングテーブルにない単語は 0 に変換される．また，CNN は固定長の入力のみを受け付けるため，全てのテキストベクトルは固定長でなければならない．そのため単語数の制限を超えるベクトルは切り詰め，制限に満たないベクトルは 0 埋めを行う．

2.2.2 ネットワークの構造

分類に用いられる 3 層のネットワークでの分類の過程について説明する．第一層は埋め込み表現層で，単語の埋め込み表現を学習する．単語の埋め込み表現は，単語の意味を表すベクトルを作成できる手法として Mikolov ら [24] や Pennington ら [25] によって提案されている．Kim の文献 [20] では埋め込み表現の学習ツールである word2vec [24] を利用して埋め込み表現の学習を行っているが，本研究では簡易化のためネットワークに埋め込み表現層を組み込み，ネットワーク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．第二層は畳み込み・プーリング層である．畳み込みでは，新しい特徴を抽出するためのフィルターを学習する．本モデルでは，n-gram 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ーを適用する．ここで用いるフィルターは，横幅が埋込表現のサイズの 128 で高さが 3, 4, 5 の 3 種類のフィルター各 128 個である．フィルター適用の結果として，計 384 個の特徴マップが得られる．次にプーリングでは，フィルタ一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を適用する．ここで

のマックスプーリングは各フィルターで得られた特徴マップの最大値を最も重要な情報として取得するため，畳み込みが適用された後の 384 個の特徴マップを 384 個のノードに変換することが出来る．第 3 層は全結合層で，出力のノードを 2 つとし，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．

2.2.3 ハイパーパラメータと訓練アルゴリズム

本研究ではネットワークのハイパーパラメータのチューニングは行っていない．ネットワークのハイパーパラメータの選択肢は膨大であり，コストがかかってしまうからである．また，層数等をチューニングすることは参考としたモデルを大きく変えてしまうことにもなる．本研究の目的は深層学習による分類器の適用可能性と性能を調査することであり，新しいモデルの設計では無いためチューニングは行わなかった．

学習におけるパラメータ更新の際の最適化アルゴリズムには Adam [26] を利用した．Adam はニューラルネットワークの学習でよく利用されてきた確率的勾配降下法 (SGD) よりも早く誤差を収束させることが出来るアルゴリズムであると報告されている．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った．

2.3 Commit Guru

不具合予測モデルの研究において実験データの公開性と透明性は重要であるとされてきた [17]．そのため，我々は Rosen らが公開している Commit Guru [15, 16] から得られるデータを利用した．Commit Guru は Kamei らの変更レベルの不具合予測モデル [6] を Web アプリケーションとして実装したものであり，予測の際に取得した変更の不具合ラベル情報とメトリクスの公開も行っている．本研究では不具合ラベルはデータのラベル付けに利用し，メトリクスはベースラインモデルで用いる．Commit Guru の機能である (1) 不具合混入コミットのラベル付と (2) メトリクスの収集について以下に述べる．

2.3.1 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，不具合を混入したコミット (buggy) とそれ以外のコミット (clean) としてラベル付けをする．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できる．始めに，Commit Guru は Hindle らの研究 [27] におけるコミットを分類するためのキーワードリストを元にコミットメッセージを解析し，'bug', 'fix', 'wrong', 'error', 'fail', 'problem', 'patch' といったワードがあれば，そのコミットを不具合を修正しているであろうコミットであるとする．次に，修正を行ったコミットから不具合を混入したであろうコミットを特定する．まず diff を用いてどの行が修正を行ったコミットによって変更されたのかを見つける．ここで変更された行に対して，バージョン管理システムの annotate/blame コマンドを用いることで，それらの修正された行がどのコミットで混入されたのかを特定する．ここで見つかった不具合修正されたソースコードを混入したコミットを，不具合を混入したであろうコミットとしてラベル付けする．また，修正コミットから不具合混入コミットを推定する有名な手法として SZZ アルゴリズム [28] があるが少し異なっている．

2.3.2 メトリクスの収集

収集されるメトリクスは Kamei らの不具合予測モデル [6] で利用されているものと同様の 13 個の変更に関するメトリクスである．13 個のメトリクスは，lines added (変更によって追加された行数)， lines deleted (変更によって削除された行数)， lines total (変更される前のコード行数)， no. subsystem (変更されたサブシステムの数)，no. directories (変更されたディレクトリの数)，no. files (変更されたファイル数)， no. developers (変更したファイルを編集したことがある人の数)，age (前回の変更からの期間)， no. unique changes (修正されたファイルに対するこれまでの変更の数)，experience (変更を行った開発者の総コミット数)， recent experience (コミットの日付で重み付けされた experience)，subsystem experience (変更されたサブシステムに対するこれまでの変更の数)，entropy (変更が多くのファイルに渡って行われているほど大きくなる値) となっている．

3. 研究の方法

本研究の方法の概要を図 3.1 に示す．本研究の主要なステップをまとめると次のようになる: (1) リポジトリから変更 (コミット) を取得し，Commit Guru から得られる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．(2) データの準備: 各変更に対し，変更されたソースコードを収集し変更レベルのソースコード片を作成する．また，Commit Guru から変更に関する 13 個のメトリクスも取得する．(4) 実験データセットの作成: 不要な変更をデータセットから取り除き，リサンプリングも行う．(5) モデルの性能の評価: 提案手法 W-CNN と比較のために用意した 6 つのベースラインモデルに対して 10 タイムズ 10 フォールドクロスバリデーションによる予測性能の評価を行い，その結果をまとめる．

ここで利用している Commit Guru のデータは Web [16] から CSV 形式で簡単にダウンロードできる．Commit Guru の CSV からは，Commit Guru で不具合予測に使われるメトリクスの値に加え，コミットハッシュ，変更が不具合を混入したどうかのラベル情報が取得できる．

3.1 対象とするプロジェクトデータ

本研究では，不具合予測モデル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のリポジトリを利用した．プロジェクトは，Hadoop，Camel，Gerrit，Osmand，CMake，Bitcoin，Gimp の 7 つであり，それぞれすでに Commit Guru 上で解析されており，13 個の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．より一般的な実験をすることを考慮に入れ，様々な分野（サーバーやアプリ）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選択した．プロジェクトの詳細を表 3.1 に示す．表 3.1 はそれぞれのプロジェクトごとに，使用されているプログラミング言語，変更の合計，不具合を混入した変更の割合を示している．変更のラベルは Commit Guru によって第 2.3.1 節で述べた方法で付けられてる．それぞれのプロジェクトは git で管理されており，オンラインで取得可能である．

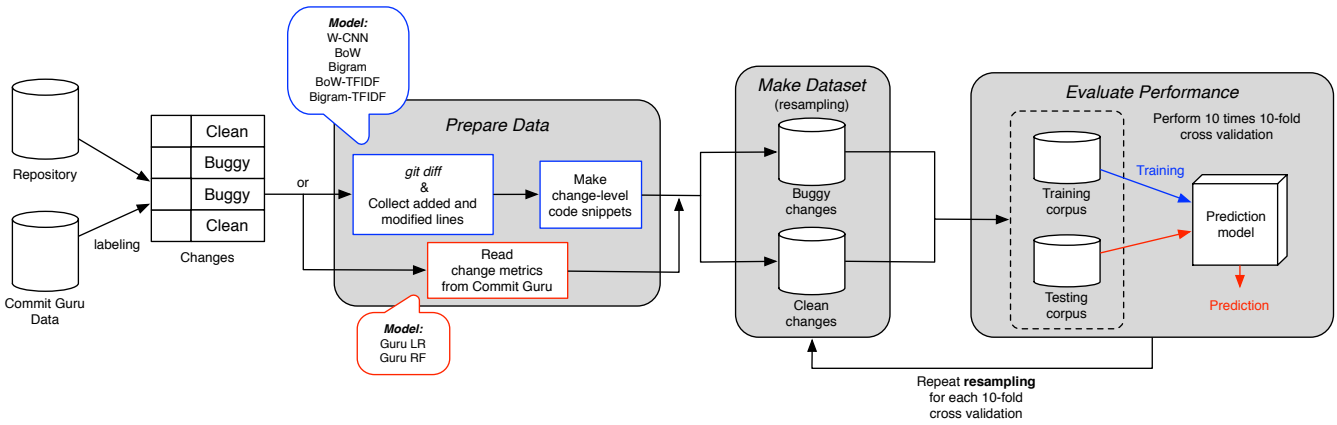


図 3.1: 研究の方法の概要

表 3.1: 対象プロジェクトの詳細

Project	Language	The total number of changes	buggy rate
Hadoop	Java	13920	24.8%
Camel	Java	24740	23.2%
Gerrit	Java	18794	20.1%
Osmand	Java	31366	14.0%
CMake	C++	28400	10.1%
Bitcoin	C++	11093	14.4%
Gimp	C++	37116	22.5%

3.2 ベースラインモデル

より正確に提案手法 W-CNN を評価するために，6 つのベースラインを用意し比較を行う．従来の変更レベルの不具合予測モデルは RQ2，従来のテキスト分類モデルは RQ3 に対応している．

3.2.1 従来の変更レベルの不具合予測モデル

我々は従来の不具合予測モデルとして，Commit Guru から得られる 13 個のメトリクスを用いることで容易に構築することができる亀井らのモデル [6] を利用した．亀井らのモデルでは，メトリクスをロジスティック回帰で学習することで予測モデルを構築していたが，本研究ではランダムフォレストも利用する．

Guru LR(Commit Guru Metrics + Logistic Regression) モデル．Guru LR モデルは，亀井らの手法と同様に変更に関するメトリクスとロジスティック回帰により構築される不具合予測モデルである．

Guru RF(Commit Guru Metrics + Random Forest) モデル．汎化能力を向上させるために，近年の不具合予測ではアンサンブル学習も用いられている．中でもよく使われているものの一つがランダムフォレストである [17]．ランダムフォレストはランダムに説明変数を選択し複数の決定木を作成することで，決定木単体よりも汎化性能を向上させたモデルである．Guru RF モデルは，ランダムフォレストにメトリクスを学習させることで構築される不具合予測モデルである．

3.2.2 従来のテキスト分類モデル

我々が提案する不具合予測モデル W-CNN は深層学習によるテキスト分類モデルに基づいている．ソースコードに潜む不具合の分類能力を比較するため，Zhang らの文献 [21] で用いられていた従来のテキスト分類モデルを参考にベースラインモデルを用意した．以下のテキスト分類モデルは特徴生成部と予測部であるロジスティック回帰で構成される．

BoW(Bag-of-Words) モデル．Bag-of-Words は単語の出現のみを特徴として考えることでテキストをベクトル化する手法である．訓練データから語彙リストを作成し，単語ごとに単語がテキストに含まれているかどうかを判定しベクトルとする．BoW モデルは，このベクトルとロジスティック回帰で構成されるテキスト分類モデルである．

Bigram モデル．Bigram は単語の出現のみでなく単語の共起も特徴として考慮したテキストのベクトル化手法である．Bigram の場合は隣り合った 2 つの単語を一つの単語列片として扱い，単語列片の出現を特徴として考える．Bigram は Bag-of-Words と同様に訓練データから単語列片リストを作成し，テキストにそれらが含まれているかどうかを判定しベクトル化する．Bigram モデルは，このベクトルとロジスティック回帰で構成されるテキスト分類モデルである．

BoW-TFIDF(Bag-of-Words とその TFIDF) モデル．このモデルでは，Bag-of-Words から得られた特徴に対して TFIDF(Term-Frequency Inverse-Document-Frequency) [29] を用いて重み付けを行う．Term-Frequency はあるテキスト中にある単語が出現する割合であり，テキスト内で頻出する単語に重みを付ける効果がある．Inverse-Document-Frequency は，全テキスト中で，ある単語が出現するテキストの割合の逆数に対数を取ったもので，多くのテキストに現れる単語の影響を小さくする効果がある．TFIDF は，計算された Term-Frequency と Inverse-Document-Frequency を Bag-of-Words で得られた特徴量に乗じることで重み付けを行いテキストをベクトル化する．BoW-TFIDF モデルは，このベクトルとロジスティック回帰で構成されるテキスト分類モデルである．

Bigram-TFIDF(Bigram とその TFIDF) モデル．BoW-TFIDF モデルと同様に，このモデルでは Bigram により得られた特徴量に TFIDF による重み付けを行いテキストをベクトル化する．Bigram-TFIDF モデルは，このベクトルとロジスティック回帰で構成されるテキスト分類モデルである．

3.3 データの準備

我々は不具合予測にテキスト分類モデルを用いるため，その入力となる変更レベルのソースコード片を作成する．また，ベースラインモデルの一つである従来の変更レベルの不具合予測モデルを用いる場合は，Commit Guru の Web [16] から得られる CSV から 13 個のメトリクスを取得する．

変更レベルのソースコード片の作成: 変更されたソースコードを取得するため，我々は `git diff` を使用した．ドキュメントファイルなどは不具合に繋がる可能性が低いと考えられるため，変更の差分はソースファイル (注 1) を対象にのみ取得した．`git diff` の出力の例を図 3.2 に示す．'-' で始まる行が削除された行，'+' で始まる行が追加または修正された行を示す．不具合を混入した変更を見つけることを目的としているため，'+' の行を対象のソースコードとして集めた．加えて，追加または修正された行の前後 3 行も文脈情報として集めた．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．変更により変更されたファイルごとに行の取得を行い，各ファイルごとに取得したソースコードをファイル名で繋げることで一つの変更レベルのソースコード片とする．変更レベルのソースコード片の例を図 3.3 に示す．このソースコード片を lscp [23] で単語分割したものを入力とし，W-CNN とベースラインモデルの一つである従来のテキスト分類モデルはそれぞれの方法で分類を行う．

3.4 実験データセットの作成

3.4.1 ソースコードを変更していない変更の削除

提案手法 W-CNN はソースコードの分類により不具合予測を行うため，ソースコードに混入される不具合のみを予測対象としている．つまり設定ファイルの記述が原因となる誤動作等は対象としていない．また，そもそもソースコードを変更しない場合は不具合混入に直結しない場合が多いと考えられる．そのため，ソースコードを全く変更していない変更や，ソースファイルのコメントのみを変更している変更は対象外として元のデータから削除した．

(注 1)：対象のソースファイルの拡張子: java, c, h, cpp, hpp, cxx, hxx

```
@@ -3062,7 +3062,9 @@
      */
     public String[] getCreateTableSQL(Table table) {
         StringBuffer buf = new StringBuffer();
-        buf.append("CREATE TABLE ").append(getFullName(table, false));
+        String tableName = checkNameLength(getFullName(table, false),
+                   maxTableNameLength, "long-table-name");
+        buf.append("CREATE TABLE ").append(tableName);
         if (supportsComments && table.hasComment()) {
             buf.append(" ");
             comment(buf, table.getComment());
```

図 3.2: git diff コマンドの出力例 (Camel Project)

```
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; }
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 3.3: 変更レベルのソースコード片の例 (Camel Project)

3.4.2 データのリサンプリング

表 3.1 に示す通りに，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，予測モデルの学習性能を下げてしまう可能性がある．不具合予測はデータのリサンプリングにより性能を上げることが可能であると知られており [9]，不具合予測モデル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 [6,7,9]．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 3.2 に元のデータからソースコードを変更していない変更を削除し，リサンプリングも実行した後のプロジェクトごとのデータ数を示す．

3.5 モデルの性能評価

3.5.1 10 タイムズ 10 フォールドクロスバリデーション

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，予測モデルの評価に 10 タイムズ 10 フォールドクロスバリデーションを利用した．10 タイムズ 10 フォールドクロスバリデーションは 10 フォールドクロスバリデーションを 10 回実行する評価手法であり，不具合予測モデルの研究でも利用されている [7]．10 フォールドクロスバリデーションではまず全データを 10 フォールドに分割する．そして 9 フォールド分をトレーニングセットとし，残りの 1 フォールド分をテストセットとした実験を全てのフォールドでテストが終わるまで繰り返す．この 10 フォールドクロスバリデーションの処理を繰り返すごとに，ランダムアンダーサンプリングを再度実行することで削除するデータを変更し，データの偏りをさらに減らした実験を行った．10 タイムズ 10 フォールドクロスバリデーションにより得られた評価指標の値の平均をモデルの評価値とする．

3.5.2 パラメータのチューニング

不具合予測モデルにおいてパラメータチューニングはその性能に大きく影響をあたえる場合がある [10]．そのため本研究ではパラメータによるモデルへの影響を考

表 3.2: リサンプリング済みのデータセット

Project	Buggy changes	Clean changes
Hadoop	3280	3280
Camel	5620	5620
Gerrit	3470	3470
Osmand	4150	4150
CMake	2790	2790
Bitcoin	1450	1450
Gimp	8080	8080

慮し，予測モデルのパラメータチューニングを行った．ロジスティック回帰ではオーバーフィッテングを避けるための正則化パラメータ C をチューニングする．このパラメータの値が小さいほど正則化は強くなり，学習データに対して強くフィッティングする．パタメータの候補値は，(0.001, 0.01, 0.1, 1, 10, 100) とした．ランダムフォレストでは決定木の数を決定するパラメータである n_estimators をチューニングする．パラメータの候補値は，(10, 20, 30, 40, 50, 60, 70, 80, 90, 100) とした．また，既存のテキスト分類手法では，Bag-of-Words 等で生成されるテキストベクトルの大きさもチューニングすべきパラメータとする．これは 1000 間隔で 5000 から 20000 までの値と特徴数の最大値をパラメータの候補値とする．W-CNN に関しては，パラメータ数が膨大でありコストがかかるためパラメータチューニングは行わない．パラメータチューニングには機械学習ツールキットである scikit-learn [11] が提供する GridSearch を利用した．チューニングは各パラメータの組み合わせごとにトレーニングデータのみで 5 フォールドクロスバリデーションを行い，得られた F1 値が最も高い組み合わせを最良のパラメータセットとする．

3.5.3 評価指標の計算

予測性能を評価するために，Recall(再現率)，Precision(適合率)，F1-score(F1 値) を利用する．これらは表 3.3 に示す混同行列から計算することが出来る．Recall は実際に不具合であったものの中で不具合と分類できたものの割合を示し，$Recall = \frac{TP}{TP+FN}$ と定義される．Precision は不具合であると分類したものの中で実際に不具合であったものの割合を示し，$Precision = \frac{TP}{TP+FP}$ と定義される．F1 値は Precision と Recall の調和平均で，$F1 = \frac{2 \times Recall \times Precision}{Recall + Precision}$ と定義される．本研究では閾値を 0.5 としてこれらの評価指標を計算する．

しかし，Recall，Precision，F1 値の値は設定した閾値に大きく依存する．そのため本研究では様々な閾値を考慮した性能を評価するために，ROC(Reciever Operating Charactristic) 曲線から求められる AUC(Area Under the Curve) も利用する．図 3.4 に ROC 曲線と AUC の例を示す．ROC 曲線は，分類器が出力した全ての確率を閾値の候補とし，その閾値ごとに得られる True Positeve 率 (TP 率) と False Positive 率 (FP 率) に対応する座標を通るようにプロットされたグラフである．図 3.4 における ROC 曲線上のドットの数は閾値の数に対応している．TP 率と FP 率も表 3.3 に示す混同

表 3.3: 混同行列

	Classified as	
True class	clean	buggy
clean	TN	FP
buggy	FN	TP

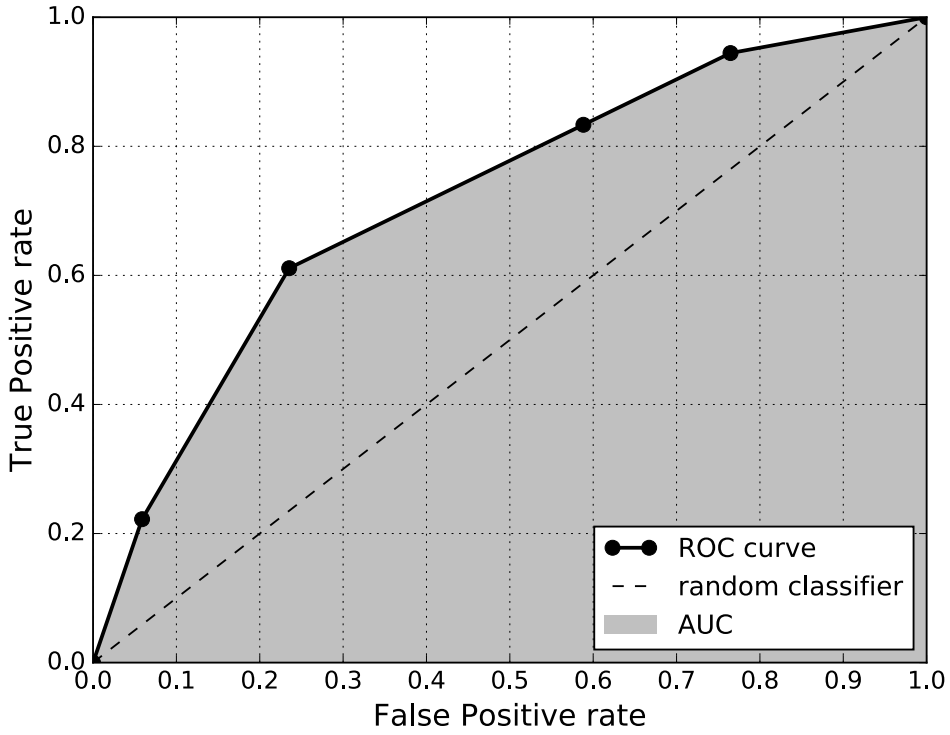


図 3.4: ROC 曲線と AUC の例

行列から計算することが出来る．TP 率はある閾値における不具合データの網羅率で TP 率 $= \frac{TP}{TP+FN}$ と定義され，FP 率はある閾値において全クリーンデータの内，不具合であると誤判定してしまったものの割合で FP 率 $= \frac{FP}{TN+FP}$ と定義される．図 3.4 に示されるように ROC 曲線は各閾値において TP 率が FP 率より高いほど上方向に広がる．そのため良い分類器である程，ROC 曲線下の面積である AUC は高くなる．AUC は 0 から 1 の値を取るが，図 3.4 の破線のようにどの閾値でも TP 率と FP 率が同等であれば AUC は 0.5 となり，ランダムな分類が取る値として分類器の性能の一つの基準となる．

4. 結果

4.1 RQ1: 提案手法 W-CNN による不具合の学習は可能か？

4.1.1 概要

提案手法 W-CNN の学習能力を調査するため，W-CNN を用いて 7 つの対象プロジェクトに対し予測実験を行い学習過程を記録した．通常，深層学習モデルは学習を進めていく程，汎化性能の向上が見込めるはずであるが，学習の過程で過剰適合が問題となる場合もある．ここでは学習性能を示す訓練誤差と汎化性能を示すテスト誤差を記録することにより，W-CNN が不具合を学習できているか，また過剰適合を起こさず汎化できているかを調べる．さらに AUC，F1 値，Precision，Recall も記録することで学習過程における分類性能も調べる．この学習過程の記録は，10 タイムズ 10 フォールドクロスバリデーションによりプロジェクトごとに 100 回ずつ行われ，その平均を最終的な記録とする．また，深層学習モデルにおいてどの程度まで学習のイテレーションを進めるかは重要なパラメータである．本研究ではイテレーションの単位をエポックとしており，予測実験では 50 エポックまでの学習過程を記録した．学習過程の結果より，どの程度のエポック数を学習したモデルの結果を RQ2 以降に利用すべきかを決定する．決定したエポック数分学習されたモデルで得られる評価指標の値を W-CNN の結果として RQ2 以降で用いる．

4.1.2 結果

学習が進むごとに訓練誤差とテスト誤差が減少していることから，学習と汎化が確認できた．図 4.1 に，各プロジェクトの学習過程における訓練誤差 (train loss)，テスト誤差 (test loss) を示す．この図からは，学習を進める度に全てのプロジェクトにおいて訓練誤差とテスト誤差が共に減少していることがわかる．訓練誤差は小さいほど訓練データに対して適合していることを示し，テスト誤差も同様に小さいほどテストデータに適合していることを示す．ただし，図 4.1 より 4/7 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とテスト誤差の値の差が大きくなり始めていることが見られる．これより 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，汎化は進み辛くなっていることがわ

かる．また，残りの 3/7 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た．これらより，エポック数 15 を学習が進んだ状態として以降の RQ で利用するエポック数の一つとする．

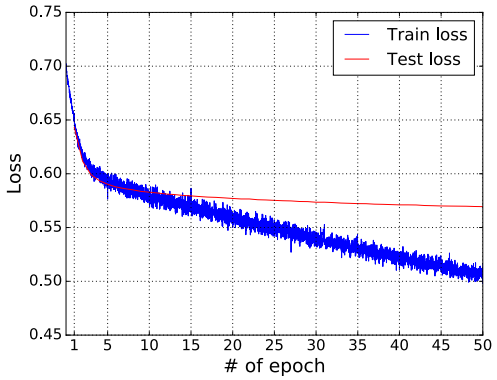
学習が進むごとに各評価指標の値が上昇していることを確認でき，最高で平均 **AUC0.817**，**F1 値 0.743** が得られた．表 4.1 は各プロジェクトでの実験で得られた評価指標の平均を 5 エポックごとにまとめたものである．それぞれの評価指標の値が学習が進むごとに上昇していることから，表 4.1 の結果からも W-CNN は学習により汎化でき，不具合を分類できていることがわかる．エポック数が増えるごとに評価指標の上昇が緩やかなものとなっているが，Recall に関しては汎化性能の上昇が緩やかになり始めるであろう 15 エポックから，50 エポックまで Precision を落とさずに 3%ポイント上昇している．タイムコストはかかってしまうが，エポック数 50 をより学習を十分に進めた状態として以降の RQ で利用する二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 は学習データに対し適合可能であり，最高で平均 AUC0.817，F1 値 0.743 が得られ不具合の分類も可能であることがわかった．また，得られた結果からエポック数 15 と 50 時点のモデルによる評価指標の結果を以降の RQ で用いることとした．

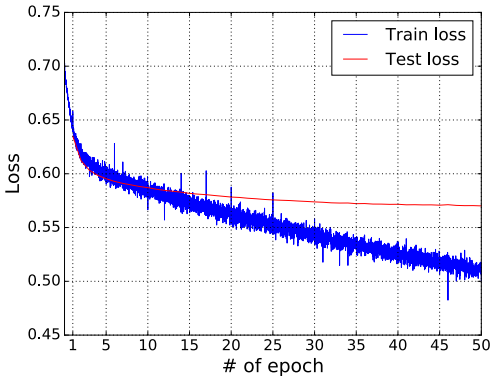
4.2 RQ2: 提案手法 W-CNN は既存の不具合予測モデルより性能が良いか？

4.2.1 概要

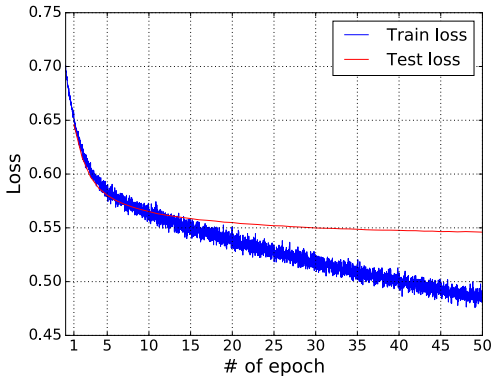
提案手法 W-CNN の有効性をより正確に評価するため，既存の不具合予測モデルを基準とした性能の比較実験を行った．ここでは，W-CNN と亀井らに提案されたモデル [6] の不具合予測モデルとしての性能の差を調べる．亀井らのモデルは変更に関するメトリクスを用いて不具合予測モデルを構築する．必要なメトリクスは Commit Guru から得られるデータから取得できるため再現が容易である．メトリクスをロジスティック回帰で学習させたモデルを Guru LR，ランダムフォレストで学習させたモデルを Guru RF とする．提案モデルとしては RQ1 より，15 エポック学習させた状態のモデル (W-CNN 15) と 50 エポック学習させた状態のモデル (W-CNN 50)



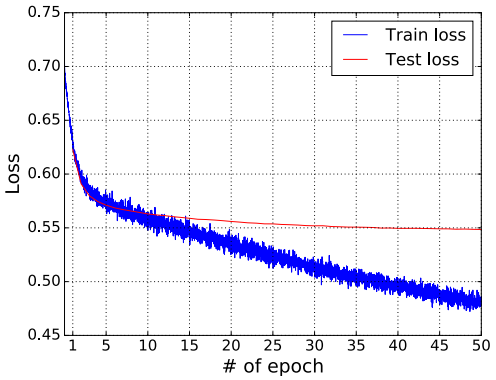
(a) Hadoop



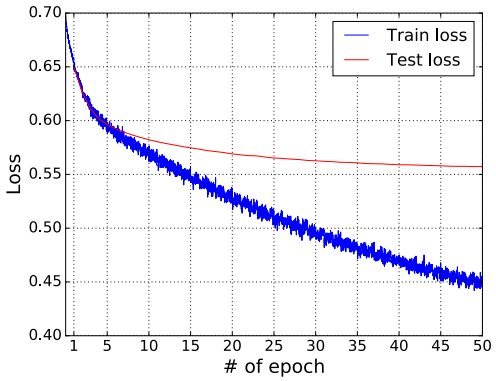
(b) Camel



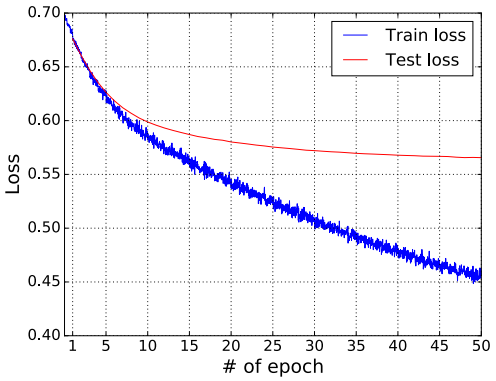
(c) Gerrit



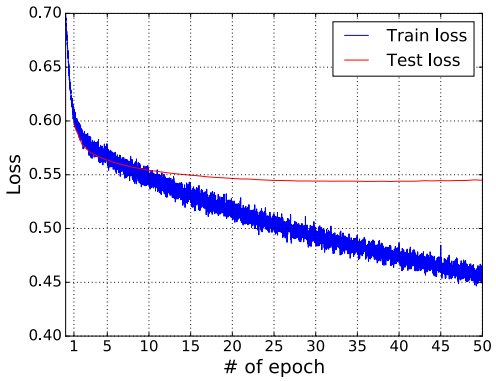
(d) Osmand



(e) CMake



(f) Bitcoin



(g) Gimp

図 4.1: 各プロジェクトの学習過程

表 4.1: 学習過程における各評価指標の平均値

the number of epoch	AUC	F1	Precision	Recall
5	0.788	0.713	0.722	0.707
10	0.797	0.721	0.729	0.715
15	0.803	0.727	0.733	0.722
20	0.807	0.731	0.736	0.729
25	0.811	0.735	0.738	0.733
30	0.813	0.738	0.737	0.740
35	0.815	0.740	0.738	0.743
40	0.816	0.741	0.737	0.747
45	0.816	0.742	0.736	0.750
50	0.817	0.743	0.736	0.752

の結果を用いる．

4.2.2 結果

より十分に学習を進めた状態の W-CNN 50 は，既存の不具合予測モデルの性能を上回った．表 4.2 は，W-CNN 15，W-CNN 50，Guru LR，Guru RF により得られた AUC，F1 値，Precision，Recall の値をまとめたものである．表 4.2 より全ての評価指標において，各プロジェクトで得られた結果の平均は W-CNN 50 が最良であったことがわかる．しかし，W-CNN 15 は既存手法で最良であった Guru RF に性能がわずかに及ばないということがわかった．このことから，W-CNN は学習を進めることで既存のモデルを超えうる性能をソースコードのみの学習から得られることがわかった．

W-CNN 50 と既存の不具合予測モデル間に統計的に有意な差は見られた．表 4.3 に，W-CNN 50 と既存の不具合予測モデルである Guru LR，Guru RF 間で行った wilcoxon 検定の p 値を示す．wilcoxon 検定は得られた 2 郡の結果に有意差があるかを検定するために用いられる検定手法である．また wilcoxon 検定はノンパラメトリック検定であり，予測結果の母集団の分布が正規分布に従うか不明である不具合予測モデルの評価に用いられている [7]．W-CNN 50 は他のモデルより結果が良いものであったが，Guru RF との差は大きいものでなかったためその有意差を調べた．本研究では有意水準を 0.05 として，p 値が 0.05 以下なら有意であると判定する．表 4.3 より，W-CNN 50 と Guru RF との間において得られた合計 28 個の結果の内，4 つに有意差が無いものがあったことがわかった．特に Gerrit プロジェクトにおいて高い p 値が集中している傾向にあり，プロジェクトにより性能の差が現れにくいものがあったことがわかった．しかし，その他大部分の結果において有意差が確認できたため，W-CNN 50 と Guru RF 間の性能の差に有意差はあるものとする．また，Guru LR との間には十分に有意差があることがわかった．

提案手法 W-CNN は学習を十分に進めた場合，既存の不具合予測モデルを超える性能を示した．また，既存の不具合予測モデルとの間に統計的有意差を確認することも出来た．ただし，プロジェクトによって性能に有意差が出にくいものがあった．

Project	W-CNN 15	W-CNN 50	Guru LR	Guru RF
Hadoop	0.788	0.802	0.733	0.798
Camel	0.783	0.798	0.725	0.781
Gerrit	0.817	0.832	0.736	0.831
Osmand	0.810	0.825	0.714	0.792
CMake	0.803	0.821	0.751	0.831
Bitcoin	0.793	0.811	0.712	0.805
Gimp	0.825	0.830	0.735	0.817
AVG	0.803	0.817	0.730	0.808

(a) AUC

Project	W-CNN 15	W-CNN 50	Guru LR	Guru RF
Hadoop	0.715	0.732	0.652	0.724
Camel	0.713	0.727	0.646	0.709
Gerrit	0.741	0.753	0.662	0.753
Osmand	0.728	0.747	0.643	0.716
CMake	0.728	0.745	0.694	0.750
Bitcoin	0.716	0.740	0.639	0.732
Gimp	0.746	0.756	0.641	0.739
AVG	0.727	0.743	0.654	0.732

(b) F1 値

Project	W-CNN 15	W-CNN 50	Guru LR	Guru RF
Hadoop	0.717	0.721	0.709	0.717
Camel	0.712	0.720	0.685	0.712
Gerrit	0.743	0.751	0.695	0.757
Osmand	0.760	0.754	0.683	0.728
CMake	0.718	0.727	0.677	0.736
Bitcoin	0.725	0.728	0.657	0.737
Gimp	0.757	0.749	0.693	0.742
AVG	0.733	0.736	0.686	0.733

(c) Precision

Project	W-CNN 15	W-CNN 50	Guru LR	Guru RF
Hadoop	0.714	0.746	0.603	0.733
Camel	0.714	0.735	0.612	0.707
Gerrit	0.740	0.756	0.632	0.749
Osmand	0.699	0.742	0.608	0.705
CMake	0.740	0.766	0.713	0.766
Bitcoin	0.710	0.753	0.622	0.727
Gimp	0.737	0.764	0.596	0.737
AVG	0.722	0.752	0.627	0.732

(d) Recall

表 4.2: 各評価指標の結果 (ハイライトはモデル間で最も良い値を示す)

Project	Guru LR	Guru RF
Hadoop	<2.2e-16	0.0006004
Camel	<2.2e-16	<2.2e-16
Gerrit	<2.2e-16	0.3577
Osmand	<2.2e-16	<2.2e-16
CMake	<2.2e-16	7.697E-08
Bitcoin	<2.2e-16	0.009672
Gimp	<2.2e-16	<2.2e-16

(a) AUC における p 値

Project	Guru LR	Guru RF
Hadoop	<2.2e-16	0.00008255
Camel	<2.2e-16	2.22E-16
Gerrit	<2.2e-16	0.7939
Osmand	<2.2e-16	<2.2e-16
CMake	<2.2e-16	0.03219
Bitcoin	<2.2e-16	0.005021
Gimp	<2.2e-16	4.441E-16

(b) F1 値における p 値

Project	Guru LR	Guru RF
Hadoop	0.00009386	0.1556
Camel	<2.2e-16	0.00009795
Gerrit	<2.2e-16	0.01564
Osmand	<2.2e-16	2.442E-15
CMake	<2.2e-16	0.001089
Bitcoin	<2.2e-16	0.02388
Gimp	<2.2e-16	0.0005286

(c) Precision における p 値

Project	Guru LR	Guru RF
Hadoop	<2.2e-16	0.0002377
Camel	<2.2e-16	3.02E-13
Gerrit	<2.2e-16	0.01105
Osmand	<2.2e-16	<2.2e-16
CMake	6.661E-16	0.7505
Bitcoin	<2.2e-16	1.255E-08
Gimp	<2.2e-16	1.332E-15

(d) Recall における p 値

表 4.3: 100 回分の評価に基づく W-CNN 50 と既存の不具合予測モデル間の有意差

4.3 RQ3: 提案手法 W-CNN は既存のテキスト分類器よりもソースコード片の分類能力が高いか？

4.3.1 概要

提案手法 W-CNN は畳み込みニューラルネットワークによるテキスト分類モデルを利用している．この研究設問では，W-CNN のソースコード内に潜む不具合の学習・分類性能をより評価するため，既存のテキスト分類モデルとの性能の比較を行う．既存のテキスト分類モデルとして，ソースコードを Bag-of-Words を利用してベクトル化する BoW モデル，Bigram を利用する Bigram モデル，Bag-of-Words とその TFIDF を利用する BoW-TFIDF モデル，Bigram とその TFIDF を利用する Bigram-TFIDF モデルを用いた．全てのモデルはロジスティック回帰により構築される．RQ2 と同様に提案モデルとして，15 エポック学習させた状態の W-CNN 15 と 50 エポック学習させた状態の W-CNN 50 の結果を用いる．

4.3.2 結果

より十分に学習を進めた状態の **W-CNN 50** は，既存のテキスト分類モデルを **AUC**，**F1 値**，**Recall** で上回った．表 4.4 に，W-CNN 15，W-CNN 50，BoW，Bigram，BoW-TFIDF，Bigram-TFIDF から得られた AUC，F1 値，Precision，Recall の結果を示す．この表より，Precision 以外では各プロジェクトで得られた結果の平均値において，より学習を進めた状態である W-CNN 50 が他のモデルを上回っていることが確認できる．このことから，W-CNN は学習を進めることで既存のテキスト分類モデルよりもソースコード内の不具合を良く学習し，より高い精度で分類可能であることがわかった．

W-CNN 50 と既存のテキスト分類モデル間に統計的に有意な差を確認できた．表 4.5 に W-CNN 50 と既存のテキスト分類モデルである BoW，Bigram，BoW-TFIDF，Bigram-TFIDF 間で行った wilcoxon 検定の p 値をまとめた．この表より，既存のテキスト分類モデルで最高の結果を記録した Bigram との差は全て有意なものであったことがわかった．しかし，W-CNN 50 と BoW 間の Precision に関する 3 つの差において有意差がなかったものがあった．これは W-CNN が Precision に関しては BoW と同等

程度の性能しか記録できなかったからであると考えられる．その他の値については有意差を十分に確認することが出来た．

提案手法 W-CNN はソースコード内の不具合を学習・分類する能力において既存のテキスト分類モデルよりも良い性能を持つことを示した．ただし Precision のみは既存のテキスト分類モデルと同等かそれ以下の結果となった．また，既存のテキスト分類モデルとの間に十分な統計的有意差を確認することも出来た．

4.4 RQ4: 提案手法 W-CNN のタイムコストはどの程度か？

4.4.1 概要

深層学習による分類モデルは，ロジスティック回帰やランダムフォレスト等よりも学習にタイムコストを必要とする．タイムコストを考慮した上で，深層学習による分類器の適用可能性を考察するため，W-CNN とベースラインとして挙げた 6 つの既存のモデルの学習にかかった時間を計測した．既存モデルに関してはパタメータチューニングも行っており，チューニングにかかる時間も学習のタイムコストに含めるようにした．また評価実験は，Intel Xeon CPU E5-1620 v3 @ 3.50GHz，NVIDIA GeForce GTX TITAN X を用いて行われている．

4.4.2 結果

W-CNN が既存のモデルを上回るためには，タイムコストをかけることが必要となる．表 4.6 は，各モデルが学習にかけたタイムコストをまとめたものである．表 4.6 より，既存のモデルと同等の性能であった W-CNN 15 でも，他のモデルと比較して学習に多くのタイムコストを要することがわかった．しかし RQ2, 3 より，W-CNN はタイムコストをかけ十分に学習を進めることにより，他のモデルを上回る程の学習・分類能力を持つこともわかっている．

既存の不具合予測モデル (**Guru LR, Guru RF**) は **10** 秒程度で学習を終えるのに対し，**W-CNN** では同等の精度にするため **5** 分程度を要した．表 4.6 より，既存モデル Guru LR は学習において 1 秒以下，Guru RF は 2 ～ 13 秒のみしか要さなかったことがわかる．これに対して既存の不具合予測モデルと同程度の性能であった W-CNN

Project	W-CNN 15	W-CNN 50	BoW	Bigram	BoW-TFIDF	Bigram-TFIDF
Hadoop	0.788	0.802	0.790	0.797	0.736	0.764
Camel	0.783	0.798	0.781	0.792	0.675	0.767
Gerrit	0.817	0.832	0.801	0.818	0.776	0.806
Osmand	0.810	0.825	0.809	0.809	0.798	0.804
CMake	0.803	0.821	0.784	0.811	0.790	0.806
Bitcoin	0.793	0.811	0.790	0.794	0.730	0.732
Gimp	0.825	0.830	0.818	0.827	0.787	0.801
AVG	0.803	0.817	0.796	0.807	0.756	0.783

(a) AUC

Project	W-CNN 15	W-CNN 50	BoW	Bigram	BoW-TFIDF	Bigram-TFIDF
Hadoop	0.715	0.732	0.715	0.715	0.693	0.699
Camel	0.713	0.727	0.704	0.707	0.676	0.696
Gerrit	0.741	0.753	0.709	0.732	0.701	0.726
Osmand	0.728	0.747	0.719	0.722	0.718	0.721
CMake	0.728	0.745	0.712	0.730	0.717	0.729
Bitcoin	0.716	0.740	0.731	0.733	0.682	0.692
Gimp	0.746	0.756	0.731	0.739	0.713	0.723
AVG	0.727	0.743	0.717	0.725	0.700	0.712

(b) F1 値

Project	W-CNN 15	W-CNN 50	BoW	Bigram	BoW-TFIDF	Bigram-TFIDF
Hadoop	0.717	0.721	0.722	0.744	0.661	0.693
Camel	0.712	0.720	0.722	0.744	0.585	0.696
Gerrit	0.743	0.751	0.747	0.771	0.695	0.734
Osmand	0.760	0.754	0.781	0.772	0.739	0.734
CMake	0.718	0.727	0.712	0.747	0.705	0.719
Bitcoin	0.725	0.728	0.694	0.671	0.607	0.588
Gimp	0.757	0.749	0.769	0.788	0.716	0.731
AVG	0.733	0.736	0.735	0.748	0.673	0.699

(c) Precision

Project	W-CNN 15	W-CNN 50	BoW	Bigram	BoW-TFIDF	Bigram-TFIDF
Hadoop	0.714	0.746	0.709	0.688	0.732	0.706
Camel	0.714	0.735	0.688	0.674	0.802	0.696
Gerrit	0.740	0.756	0.676	0.697	0.713	0.718
Osmand	0.699	0.742	0.667	0.678	0.698	0.710
CMake	0.740	0.766	0.713	0.716	0.730	0.740
Bitcoin	0.710	0.753	0.773	0.810	0.813	0.840
Gimp	0.737	0.764	0.698	0.695	0.710	0.716
AVG	0.722	0.752	0.703	0.708	0.742	0.732

(d) Recall

表 4.4: 各評価指標の結果 (ハイライトはモデル間で最も良い値を示す)

Project	BoW	Bigram	BoW-TFIDF	Bigram-TFIDF
Hadoop	<2.2e-16	0.00001343	<2.2e-16	<2.2e-16
Camel	<2.2e-16	3.089E-07	<2.2e-16	<2.2e-16
Gerrit	<2.2e-16	4.441E-16	<2.2e-16	<2.2e-16
Osmand	<2.2e-16	<2.2e-16	<2.2e-16	<2.2e-16
CMake	<2.2e-16	0.000004146	<2.2e-16	1.11E-15
Bitcoin	<2.2e-16	4.441E-15	<2.2e-16	<2.2e-16
Gimp	2.22E-16	0.001025	<2.2e-16	<2.2e-16

(a) AUC における p 値

Project	BoW	Bigram	BoW-TFIDF	Bigram-TFIDF
Hadoop	5.551E-15	1.31E-14	<2.2e-16	<2.2e-16
Camel	<2.2e-16	<2.2e-16	<2.2e-16	<2.2e-16
Gerrit	<2.2e-16	1.11E-15	<2.2e-16	<2.2e-16
Osmand	<2.2e-16	<2.2e-16	<2.2e-16	<2.2e-16
CMake	<2.2e-16	3.451E-13	<2.2e-16	1.278E-12
Bitcoin	0.000213	0.006074	<2.2e-16	<2.2e-16
Gimp	<2.2e-16	<2.2e-16	<2.2e-16	<2.2e-16

(b) F1 値における p 値

Project	BoW	Bigram	BoW-TFIDF	Bigram-TFIDF
Hadoop	0.317	2.153E-15	<2.2e-16	4.441E-16
Camel	0.288	5.335E-15	<2.2e-16	<2.2e-16
Gerrit	0.1526	2.418E-14	<2.2e-16	1.435E-09
Osmand	<2.2e-16	6.265E-14	5.699E-12	6.883E-15
CMake	4.811E-07	6.516E-07	7.994E-15	0.0009524
Bitcoin	6.661E-16	<2.2e-16	<2.2e-16	<2.2e-16
Gimp	1.586E-15	<2.2e-16	<2.2e-16	1.332E-15

(c) Precision における p 値

Project	BoW	Bigram	BoW-TFIDF	Bigram-TFIDF
Hadoop	3.642E-14	<2.2e-16	0.0103	3.531E-13
Camel	<2.2e-16	<2.2e-16	<2.2e-16	<2.2e-16
Gerrit	<2.2e-16	<2.2e-16	2.846E-09	3.775E-15
Osmand	<2.2e-16	<2.2e-16	<2.2e-16	4.441E-16
CMake	<2.2e-16	<2.2e-16	3.109E-15	6.388E-13
Bitcoin	7.773E-07	<2.2e-16	9.277E-07	<2.2e-16
Gimp	<2.2e-16	<2.2e-16	<2.2e-16	<2.2e-16

(d) Recall における p 値

表 4.5: 100 回分の評価に基づく W-CNN 50 と既存のテキスト分類モデル間の有意差

15 は，平均で学習に 5 分程度を要する結果となった．ただし，Guru LR，Guru RF，は合計 13 個のメトリクスを利用している．今回は Commit Guru からメトリクスを取得したため取得にかかるコストは不明であるが，メトリクスの計算，解析コストは存在するはずである．提案手法である W-CNN はソースコードのみで分類を行うためメトリクスの収集という点ではコストが低いと考えられる．

提案手法 W-CNN は，ベースラインとした既存のモデルよりも学習に多くの時間がかかるモデルであった．しかし W-CNN には，入力がソースコード片のみでモデル構築準備が容易である点や，タイムコストをかけることで既存のモデルの性能を上回ることができるという利点がある．

表 4.6: 各モデルが学習に要したタイムコスト (s)

Project	W-CNN 15	W-CNN 50	Guru LR	Guru RF	BoW	Bigram	BoW-TFIDF	Bigram-TFIDF
Hadoop	260.5	625.4	0.4	5.4	139.7	199.2	80.7	144.5
Camel	363.9	1006.2	0.5	8.6	273.9	287.2	103.1	173.0
Gerrit	300.9	692.8	0.4	5.3	125.8	160.6	57.0	100.3
Osmand	350.5	830.0	0.4	6.2	156.8	208.1	65.1	112.7
CMake	263.0	588.7	0.6	6.8	56.4	72.3	35.2	53.9
Bitcoin	219.0	384.1	0.3	2.9	30.7	53.2	22.7	36.0
Gimp	502.6	1439.6	0.9	13.5	381.5	396.9	150.3	413.5
AVG	322.9	795.3	0.5	6.9	166.4	196.8	73.4	147.7

5. 妥当性の検証

5.1 構成概念妥当性

構成概念妥当性では，実験での評価方法の妥当性について述べる．我々は評価指標として AUC，Precision，Recall，F1 値を利用した．これらは不具合予測モデルの評価指標としてよく利用されており [1–10]，本実験におけるモデルの評価指標として妥当であると考えられる．また本実験では，データの選択の偏りを取り除き，さらに統計的評価を実行するため，10 タイムズ 10 フォールドクロスバリデーションを利用した．この手法も，より正確にモデルを評価するため不具合予測モデルの評価に利用されてきている [7].

5.2 外的妥当性

外的妥当性では，実験結果の一般性についての妥当性について述べる．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た．選択したプロジェクトは，C と Java の 2 つの言語と様々な分野 (サーバー，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，世にある全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だろう．また，本実験の対象プロジェクトは全てオープンソースソフトウェアである．産業との共同研究による妥当性の向上は必要であるだろう．

5.3 内的妥当性

内的妥当性では，実験の方法の妥当性について述べる．

データへのラベリングの正確さ: 本実験で利用したデータのラベルは Commit Guru [15, 16] によりラベリングされている．しかし，Commit Guru のラベリング手法による不具合混入コミットの網羅率は完全ではない．例えば Commit Guru は不具合の特定にコミットメッセージを利用しているが，コミットメッセージにキーワード

('bug', 'fix', 'wrong' 等) が入っていなかった場合，不具合と関係していたとしてもそれを見逃してしまう．また同様のアルゴリズムとして SZZ アルゴリズム [28] があるが，SZZ アルゴリズムは不具合の特定にバグトラッキングシステムも利用するため，Commit Guru の手法よりも高い適合率で不具合の特定が可能であると考えられる．ただしどちらの手法もコミットメッセージ等の開発者からの情報の正確さに大きく依存する．

プログラムの正確さ: 実験に利用したプログラムに対してのダブルチェックは行ったが，我々が気づいていない不具合が存在している可能性がある．

6. 結言

これまで不具合予測モデルに関して様々な研究が行われてきた．最近ではより良い性能を求めて不具合予測にも深層学習が利用され始めている．しかし，我々の知る限り深層学習による分類器を不具合予測モデルとして利用した研究は行われていない．これに対し本論文では，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モデル W-CNN を提案した．また，7 つのソフトウェアプロジェクトと 6 つのベースラインモデルを用意して提案モデルの評価実験を行った．実験から得られた知見を以下に纏める．

- 提案手法 W-CNN の学習過程を記録した結果，W-CNN は学習データに対し適合可能であり，最高で平均 AUC0.817，F1 値 0.743 が得られ不具合の分類も可能であることがわかった．
- 提案手法 W-CNN は学習を十分に進めた場合，既存の不具合予測モデルを超える性能を示した．また，既存の不具合予測モデルとの間に統計的有意差を確認することも出来た．ただし，プロジェクトによって性能に有意差が出にくいものがあった．
- 提案手法 W-CNN はソースコード内の不具合を学習・分類する能力において既存のテキスト分類モデルよりも良い性能を持つことを示した．ただし Precision のみは既存のテキスト分類モデルと同等かそれ以下の結果となった．また，既存のテキスト分類モデルとの間に十分な統計的有意差を確認することも出来た．
- 提案手法 W-CNN は，ベースラインとした既存のモデルよりも学習に多くの時間がかかるモデルであった．しかし W-CNN には，入力がソースコード片のみでモデル構築準備が容易である点や，タイムコストをかけることで既存のモデルの性能を上回ることができるという利点がある．

以上の結果から深層学習による分類器は不具合予測に適用可能であると考えられる．また W-CNN は既存のモデルと比較しても劣ることは無かったことから，深層学習による分類器の適用は，不具合予測モデルを向上させるアプローチとして期待できるものの一つであると思われる．深層学習は学習にタイムコストがかかってしまう欠点があるが，今後のコンピュータの性能の向上によりこの欠点は無視できる

ようになる可能性もある．また，深層学習については現在も研究が進んでおり，今後より良い分類モデルが提案されることで不具合予測モデルのさらなる性能向上が期待できるだろう．

謝辞

本研究を行うにあたり，研究課題の設定や研究に対する姿勢，本論文の作成に至るまで，全ての面で丁寧なご指導を頂きました，本学情報工学・人間科学系水野修教授に厚く御礼申し上げます．また研究の過程で多くの貴重な助言を頂きました，Queen’s University の Professor Ahmed E. Hassan と Dr. Cor-Paul Bezemer，本学ソフトウェア工学研究室の皆さん，学生生活を通じて筆者の支えとなった家族や友人に深く感謝致します．

参考文献

[1] T. Zimmermann, R. Premrai, and A. Zeller, “Predicting defects for eclipse,” Proc. of 3rd International Workshop on Predictor models in Software Engineering, pp.1–7, 2007.

[2] S. Kim, E.J. Whitehead Jr., and Y. Zhang, “Classifying software changes: Clean or buggy?,” IEEE Trans. on Software Engineering, vol.34, no.2, pp.181–196, March/April 2008.

[3] O. Mizuno and T. Kikuno, “Training on errors experiment to detect fault-prone software modules by spam filter,” Proc. of the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE’2007), pp.405–414, Sept. 2007. Dubrovnik, Croatia.

[4] L. Aversano, L. Cerulo, and C.D. Grosso, “Learning from bug-introducing changes to prevent fault prone code,” Proc. of 9th International workshop on Principles of software evolution (IWPSE’2007), pp.19–26, ACM, New York, NY, USA, 2007.

[5] T. Jiang, L. Tan, and S. Kim, “Personalized defect prediction,” Proc. of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE’2013), pp.279–289, 2013.

[6] Y. Kamei, E. Shihab, B. Adams, A.E. Hassan, A. Mockus, A. Sinha, and N. Ubayashi, “A large-scale empirical study of just-in-time quality assurance,” IEEE Trans. Softw. Eng., vol.39, no.6, pp.757–770, June 2013.

[7] X. Yang, D. Lo, X. Xia, Y. Zhang, and J. Sun, “Deep learning for just-in-time defect prediction,” Proc. of the 2015 IEEE International Conference on Software Quality, Reliability and Security (QRS’2015), pp.17–26, 2015.

[8] S. Wang, T. Liu, and L. Tan, “Automatically learning semantic features for defect prediction,” Proc. of the 38th International Conference on Software Engineering (ICSE’2016), pp.297–308, 2016.

[9] M. Tan, L. Tan, S. Dara, and C. Mayeux, “Online defect prediction for imbalanced data,” Proc. of the 37th International Conference on Software Engineering (ICSE’2015), pp.99–108, 2015.

[10] C. Tantithamthavorn, S. McIntosh, A.E. Hassan, and K. Matsumoto, “Automated parameter optimization of classification techniques for defect prediction models,” Proc. of the International Conference on Software Engineering (ICSE’2016), pp.321–332, 2016.

[11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and douardDuchesnay, “Scikit-learn: Machine learning in python,” Journal of Machine Learning Research, vol.12, pp.2825–2830, 2011.

[12] R. Ihaka and R. Gentleman, “R: A language for data analysis and graphics,” Journal of Computational and Graphical Statistics, vol.5, no.3, pp.299–314, 1996.

[13] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015. Software available from tensorflow.org.

[14] T. Menzies, B. Caglayan, E. Kocaguneli, J. Krall, F. Peters, and B. Turhan, “The promise repository of empirical software engineering data,” 2012.

[15] C. Rosen, B. Grawi, and E. Shihab, “Commit guru: Analytics and risk prediction of software commits,” Proc. of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE’2015), pp.966–969, ACM, 2015.

[16] Commit Guru, （オンライン）, 入手先〈http://commit.guru/〉.

[17] Y. Kamei and E. Shihab, “Defect prediction: Accomplishments and future challenges,” Proc. of Leaders of Tomorrow / Future of Software Engineering Track

at International Conference on Software Analysis, Evolution, and Reengineering (SANER'2016), pp.33–45, 2016.

[18] A. Krizhevsky, I. Sutskever, and G.E. Hinton, "Imagenet classification with deep convolutional neural networks," Proc. of 26th Annual Conference on Neural Information Processing Systems 2012 (NIPS'2012), pp.1106–1114, 2012.

[19] C.D. Santos and M. Gatti, "Deep convolutional neural networks for sentiment analysis of short texts," Proc. of the 25th International Conference on Computational Linguistics (COLING'2014), pp.69–78, 2014.

[20] Y. Kim, "Convolutional neural networks for sentence classification," Proc. of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP'2014), pp.1746–1751, 2014.

[21] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," Proc. of the 28th International Conference on Neural Information Processing Systems (NIPS'2015), pp.649–657, 2015.

[22] A.N. Lam, A.T. Nguyen, H.A. Nguyen, and T.N. Nguyen, "Combining deep learning with information retrieval to localize buggy files for bug reports," Proc. of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE'2015), pp.476–481, 2015.

[23] S.W. Thomas, lscp: A lightweight source code preprocesser, GitHub（オンライン), 入手先〈https://github.com/doofuslarge/lscp〉（参照 2016-2-6).

[24] T. Mikolov, word2vec - Tool for computing continuous distributed representations of words., Google Code（オンライン), 入手先〈https://code.google.com/archive/p/word2vec/〉（参照 2016-2-6).

[25] J. Pennington, R. Socher, and C.D. Manning, "Glove: Global vectors for word representation," Proc. of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP'2014), pp.1532–1543, 2014.

[26] D.P. Kingma and J. Ba, "Adam: A method for stochastic optimization," `http://arxiv.org/abs/1412.6980`, 2014.

[27] A. Hindle, D.M. German, and R. Holt, "What do large commits tell us?: A taxonomical study of large commits," Proc. of International workshop on Mining software repositories (MSR'2008), pp.99–108, 2008.

[28] J. Śliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?," Proc. of the 2005 international workshop on Mining software repositories (MSR'2005), pp.1–5, ACM, New York, NY, USA, 2005. St. Louis, Missouri.

[29] K.S. Jones, "A statistical interpretation of term specificity and its application in retrieval," Journal of Documentation, vol.28, pp.11–21, 1972.