Vol.2020-SE-204 No.3
2020/3/2

# ソースコード片を用いた深層学習による業務ソフトウェアの不具合予測

中庭 貴洋[1]　上田 賀一[1]

**概要**：ソフトウェアの品質保証活動の効率化を支援する手法として深層学習を用いたソフトウェア不具合予測がある．本研究では医療カルテシステムを対象として学習モデルを構築，不具合予測を行いその精度を評価した．バグ修正及び仕様変更されたソースコード片の 2 分類，変更履歴が存在しないソースコード片を加えた 3 分類でモデルを構築し予測した．CNN が分類した判断根拠を可視化する Grad-CAM 法を用いて，判断根拠となるトークンを抽出した．両モデルにおいて一定の精度で不具合予測が可能であることが示された．また，Grad-CAM 法により判断基準となった単語を抽出し，分類ごとの傾向を調査した．

**キーワード**：ソフトウェア不具合予測，深層学習，バグ修正，仕様変更

# Defect Prediction of Business Software by Deep Learning Using Source Code Fragments

Takahiro Nakaniwa[1]　Yoshikazu Ueda[1]

***Abstract:*** There is software defect prediction method using machine learning to support the efficiency of software quality assurance activities. In this study, we constructed a learning model for a medical chart system, predicted defects and evaluated the accuracy. We constructed two learning models based on two classifications of source code fragments with bug fixes and specification changes, and three classifications with source code fragments without change logs. And, we predicted the defects using those models. It was shown that both models can predict defects with a certain degree of accuracy. In addition, we extracted words as basis for the decision using the Grad-CAM method, and investigated the tendency of each classification.

***Keywords:*** software defect prediction, deep learning, bug fixes, specification changes

## 1. はじめに

一つのソフトウェアを長期間にわたって運用するにあたって品質を保つことは重要である．ソフトウェアの品質を保証するためには, テストやレビュー, 修正を行うことが必要であるが, 品質保証を行っていくにも時間的, 人的コストがかかってくるため, 効率良く品質保証活動を行なうことが求められてくる．また、ソフトウェアが複雑化するにつれて人手で不具合の発生箇所, 修正箇所を短時間で特定するには難くなってくるという問題が発生してくる．そういった品質保証活動の効率化を支援する手法の一つとしてソフトウェア不具合予測がある．不具合予測とはソフトウェアの開発履歴情報やソースコードから得られる情報を用いて不具合が発生するであろう箇所を予測する手法である. 不具合箇所を予測することで, 早期での不具合発見, 修正が行えるほか, レビュー, テストの効率を改善できる. 不具合予測を行う手法の多くでは機械学習が用いられている．

機械学習を用いた手法のほか, 深層学習を用いて不具合予測を行う手法も提案されている．深層学習を用いた手法のうち, 近藤らの研究 [1] では, 変更履歴情報から得られた追加, 削除, 変更されたソースコード片のみを用いて不具合予測を行っており, 高い精度での予測を示している. また,

[1] 茨城大学
Ibaraki University

川田らの研究 [2] の中で, 近藤らの提案する不具合予測手法を用いて Java で書かれた社内開発ソフトウェアに対して不具合予測を行っており, 社内開発プロジェクトのみで構築したモデルでの不具合予測を行ったところ, オープンソースソフトウェアのみでモデルを構築, 不具合予測を行ったものと同程度の精度が得られている. これらの研究で予測を行う対象として用いられているプロジェクトは Java, C++で書かれたもののみである.

本研究では，VB.net で書かれた業務プロジェクトを対象として, 近藤らの手法を適用して不具合予測を行った. ソースコード片の情報としてバグ修正されたもの, 仕様変更で修正されたもの, 変更履歴の存在しないものの 3 つを用いて, 予測結果をバグ修正, 仕様変更の 2 分類で予測を行うほか, バグ修正, 仕様変更, 修正履歴なしの 3 分類で予測を行い, その予測精度を評価した. また，Grad-CAM 法 [3] を用いて深層学習が不具合予測を行う過程でどのトークンに注目しているかを可視化し, 分類ごとに傾向を考察した.

## 2. 関連研究

### 2.1 W-CNN

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) を用いて不具合予測を行っている. この研究で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 を用いた手法を提案していている [1]. 近藤らはこの提案手法を W-CNN(Word-CNN) と呼んでいる.

W-CNN モデルの構築は Kim のモデル [4] を参考に構築している. この手法では変更のソースコード片のみを用いて深層学習を適用しているためメトリクスに比べデータ収集が容易である, 変更ソースコード片を用いるためソースコードを作成したタイミングで不具合混入の可能性の判別を行えるといった利点がある. メトリクスを対象として深層学習を用いている Deeper [5] との比較実験を行っている. 実験結果より，Deeper に比べ W-CNN の方が学習に時間がかかるが, 予測精度が良いと報告されている.

W-CNN はソースコード片を取り出す前処理部とソースコード片の特徴から分類を行うネットワーク部に分かれている. 2.1.1 節， 2.1.2 節でそれぞれの処理の流れを説明する.

#### 2.1.1 前処理部

前処理部では予測対象のソフトウェアのコミット履歴から変更されたソースコード片の単語分割を行い, それぞれのコミットの特徴としている. 主要な処理の流れを以下に示す.

(1) ソースコード片作成

(2) 単語分割

(3) マッピング, ベクトル変換

ソースコード片作成では, 予測対象のコミット履歴から変更された差分のソースコードをつなぎ合わせた文字列を抽出する. これをソースコード片と呼ぶ. 取得する差分はソースコードファイルのみとしている. 不具合が混入したコミットを見つけるために, コミットごとにソースコードとして追加, 修正された行を収集する. さらに, 文脈行として追加, 修正された行のコメント行を除く前後 3 行も合わせて収集する. 文脈行を用いる理由として, 変更されたソースコード片のみを使う場合より, その前後行を追加することで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いう仮説を立て, 文脈行を加え不具合予測を行ったところ, 前後 3 行を加えたソースコード片での予測が精度が最も優れていたため文脈行を 3 としている.

このようにして取得したソースコードをファイルごとに収集し, それぞれのソースコードの先頭にファイル名を追加した文字列をソースコード片として扱う. ファイル名を追加する理由として, そのソースコードが持つ機能を最も端的に示すことができる文字列と考えているためである. ソースコードを収集する際, 削除行そのものは収集対象せず, 文脈行のみを考慮する.

単語分割では,W-CNN は入力文書が単語ごとに区切られている必要があるため, 作成したソースコード片を単語ごとに分割し, プログラミング言語の識別子, 予約語のみを抽出している.

また，CNN への入力は固定長でなければならないため, ソースコード片の長さを固定長にする必要がある. 固定長に満たないソースコード片の場合, 足りない文字数を 0 で埋めて固定長にする. 固定長を超えるソースコード片の場合は, 閾値以降の単語を削除し固定長にしている.

マッピング, ベクトル変換では, 固定長に揃えたソースコード片に現れる全ての単語に対して頻出順に 1 から数字を割り振り, 一つのマッピングテーブルを作成する. 作成したマッピングテーブルを利用して, ソースコード片をテキストベクトルへと変換する. マッピングテーブルにない単語は 0 とする. この処理でソースコード片を持つコミットは 1 次元の数値ベクトルへ変換される. ベクトルの各数値はソースコード片に現れる各単語に対応している.

#### 2.1.2 ネットワーク部

前処理によって得られたテキストベクトルを各コミットの特徴として，W-CNN の入力としている. ネットワーク部では特徴の抽出, 分類が CNN によって行われる. 以下にネットワーク層を示す.

(1) 埋め込み表現層

(2) 畳み込み・プーリング層

(3) 全結合層

埋め込み表現層では入力したソースコード片に存在する,

全ての単語の埋め込み表現を学習する. 埋め込み表現では各単語を表現するベクトルを求めることである．W-CNN では単語を 128 次元のベクトルへ変換するためのルックアップテーブルを作成している. この 128 が埋め込み表現のベクトルサイズである.

畳み込み層ではソースコード片から新しい特徴を抽出するためのフィルタを学習している．W-CNN では複数の単語の共起を考慮し, 横幅が埋め込み表現のベクトルサイズである 128, 縦幅が 3，4，5 の 3 種類のフィルタを各種 128 個, 総数 384 個のフィルタを持つ. それぞれのフィルタで畳み込みを行うことで 384 個の特徴マップが得られる.

プーリング層では, 畳み込み層で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し, 各特徴マップの最大値を取り出し 384 個のノードに変換している.

全結合層ではプーリング層から得られたノードを, 出力ノード数を分類するクラスの数として全結合を行う. その後, ソフトマックス関数を用いてそれぞれの分類に所属する確率を求め, ネットワークの出力としている.

## 3. 実装

本研究では,W-CNN を参考にネットワークモデルを構築した不具合予測を行う. 適用にあたって, ソースコード片の抽出ツール, 前処理部, ネットワーク部を実装した. 以下, それぞれを説明する.

### 3.1 ソースコード片の抽出

ここでは, 訓練データ, テストデータとなるソースコード片を抽出する. 抽出の対象は変更履歴が存在するソースコード片, 変更履歴の存在しないソースコード片である. 以下に, それぞれのソースコード片の概要, 抽出処理の流れを説明する.

#### 3.1.1 変更履歴が存在するソースコード片の抽出

抽出するソースコード片はバグ修正により変更されたソースコード片, 仕様変更により変更されたソースコード片である. また, 変更箇所の前後 3 行も抽出対象としている. 前後 3 行を抽出対象としている理由は 2.1.1 節のソースコード片作成で述べた, 文脈行を考慮した不具合予測を行うためである.

ソースコード中から変更箇所を抽出する処理の流れを説明する. 本研究の対象プロジェクトでは, 修正履歴情報を ID で管理しており, プログラム中で ID の文字列を検索することで修正箇所を発見できる. 対象としている業務ソフトウェアのソースコードを一部抜粋したものを以下に示す.

変更箇所は以下のような形式のコメント行で挟まれた行である.

(1) 修正方法（追加, 削除）

表 1 ID 件数と抽出ファイル数

| 修正内容 | ID 件数 | 抽出データ数 |
|---|---|---|
| バグ修正 | 610 | 1,037 |
| 仕様変更 | 484 | 1,810 |

表 2 修正履歴が存在しないソースコード片の抽出ファイル数

| | 抽出データ数 |
|---|---|
| 修正履歴なし | 6,094 |

(2) 修正内容

(3) ID

(4) 修正始めの行, 終わりの行（STA,END）

変更履歴が存在するソースコード片を自動的に抽出するにはコメント行を利用する必要がある. そのため, 修正 ID が示されたコメントを利用したソースコード片抽出ツールを作成した. ツール作成にあたって Roslyn [6] を利用した. Roslyn を利用することで，VB.Net で書かれたプログラムに対して字句解析, 構文解析を行うことが可能になり, コメント行を抽出することができる. コメント行の抽出によって, 変更履歴が存在するソースコード片の抽出を行う. そのため，ID の検索，“STA”を含むコメントと“END”を含むコメントで囲われた範囲を抽出範囲として特定できる. 抽出範囲ごとにファイルを作成し, 先頭にそのコード片が所属するクラス名を付与することでソースコード片としている. 表 1 にそれぞれの ID 件数と抽出されたデータ数を示す.

#### 3.1.2 変更履歴が存在しないソースコード片の抽出

3.1.1 節で用いたツールの処理に変更を加えたものを用いて抽出する. 3.1.1 節では“STA”を含むコメントと“END”を含むコメントで囲われた範囲を抽出範囲としていたが, 変更履歴が存在しないソースコード片を抽出する際には，“STA”と“END”で囲まれていない範囲を抽出範囲とする必要がある. また, 抽出するソースコードの行数 30 行以内とした. これは，“STA”と“END”で囲まれていない範囲を抽出すると変更履歴の存在するソースコード片に比べ, 非常に行数が大きくなってしまう. そこで変更履歴の存在するソースコード片の平均行数を調査したところ, 約 25 行であることがわかった. この行数と大きく変わらないようにソースコード片を抽出するために 30 行という制約を設けて抽出した. 表 2 に抽出されたデータ数を示す.

### 3.2 前処理

前処理では前節で抽出したソースコード片を CNN で処理可能な形式へ変換する処理を行っている．W-CNN の前処理部を参考に実装を行ったが, ソースコード片作成に関して, 変更ソースコードをファイルごとにまとめず, コメントで囲まれた範囲ごとにまとめるといった変更を行った.

ファイルごとではなくコメントの範囲ごとにまとめてい

るのは, 変更履歴情報から得られる学習データの量が理由である. 本研究では業務プロジェクトのデータのみから不具合予測を行いたい. 変更履歴から得られる抽出ファイルが少ないと学習モデルの構築が難しくなってしまう恐れがあるので，1 つの変更情報から多くの情報を作成することを考え, ファイルごとではなくコメント範囲ごとのソースコードを抽出データとしている. また, 文字数が設定した固定長より短い場合に行う 0 埋めの方式を, ソースコードの単語列が中央に来るように, 単語列の前後に 0 を埋める方式を使っている.

### 3.3 ネットワーク

W-CNN を参考に, 埋め込み表現層, 畳み込み層, プーリング層, 全結合層を持つネットワークを実装した. ネットワークの実装には，Tensorflow [7] および Keras [8] を用いた. また, 近藤らの手法を参考にハイパーパラメータの設定を行なった. 最適化アルゴリズムには Adam を利用し, 正規化項として L2 ノルムを用いた. 学習手法としてミニバッチサイズが 64 であるミニバッチ学習を実装した.

## 4. 実験

評価実験では, 主要なものとして以下 3 つの実験を行う.

実験 1　業務プログラムのみでの学習モデル構築

実験 2　バグあり, バグなしでの分類精度の評価

実験 3　バグあり, バグなし, 修正なしでの分類精度の評価

目的として, 実験 1 では実装の段階で得られた業務ソフトウェアのソースコード片のみで W-CNN が不具合予測を行えるかを調査, 実験 2, 3 では, それぞれの分類モデルでの精度を交差検証を用いて評価する.

### 4.1 学習モデルの作成

前述した関連研究では，VB.net で書かれたプロジェクトに対して不具合予測を行った実験は行っていない. そこでこの実験では VB.net で書かれた業務プロジェクトのみのソースコード片のみで W-CNN が不具合予測を行うことが可能であるか, また学習モデルの汎化性能を調査し, 実験 2, 3 で用いるエポック数を決定する.

評価するモデルとして，2 分類を行うモデルと 3 分類を行うモデルを構築, 評価する. 汎化性能を計る指標として訓練データとテストデータでそれぞれ正答率 (accuracy), 誤差 (loss) を用いた. 誤差の計算には交差エントロピーを用いている. それぞれ 100 エポック学習させ, 汎化性能を調査する. 学習の過程で, テストデータでの正答率が低くなる, または誤差が大きくなる場合には過学習を起こしていると判断する.

**表 3**　2 分類に用いる均一, 不均一データ数

| | 均一データ数 | 不均一データ数 |
|---|---|---|
| バグ修正 | 1,037 | 1,037 |
| 仕様変更 | 1,037 | 1,810 |
| 合計 | 2,074 | 2,847 |

**表 4**　3 分類に用いる均一, 不均一データ数

| | 均一データ数 | 不均一データ数 |
|---|---|---|
| バグ修正 | 1,037 | 1,037 |
| 仕様変更 | 1,037 | 1,810 |
| 変更なし | 1,037 | 2,000 |
| 合計 | 3,111 | 4,847 |

また, 表 1 よりバグ修正と仕様変更のデータ数が不均一であることがわかる. 不均一であると予測精度に影響を及ぼす可能性がある一方, データが少なすぎても精度に影響がおよぶ恐れがある. そこで, データが均一であるモデルと不均一であるモデルをそれぞれ構築し, 予測精度を比較する.

#### 4.1.1 2 分類モデル構築

バグ修正と仕様変更のソースコード片を用いて学習を行う. ここでは均一データにするため, 仕様変更のデータ数をバグ修正のデータ数に揃える. 表 3 にモデル構築に用いたデータの内訳を示す.

均一モデルの学習過程における訓練データの正答率, テストデータの正答率, 訓練誤差, テスト誤差のグラフを図 1 に示す. また, 不均一モデルの学習過程における訓練データの正答率, テストデータの正答率, 訓練誤差, テスト誤差のグラフを図 2 に示す.

両モデルとも, 学習が進むにつれて誤差が小さくなり, 正答率が大きくなっていることから，2 分類での業務ソフトウェアのみでの学習は可能であると言える.

また, 均一, 不均一モデルのどちらの訓練, テストの正答率も 20 エポックを境に値の変化が小さくなり始めていることがわかる. また, 誤差も同様に 20 エポックを境に訓練, テスト誤差の変化が小さくなり始めており, これ以降のエポックでは汎化が進みにくくなっている. よって，20 エポック時点での学習モデルを十分に汎化性能を持ったモデルと考え，4.2 節での実験ではエポック数 20 を用いて評価を行っていく.

均一, 不均一モデルの違いとしてグラフから読み取れることとして, 均一モデルと比較して不均一モデルでのテスト正答率が高く, また誤差が小さいことが分かる.

#### 4.1.2 3 分類モデル構築

バグ修正と仕様変更, 修正なしのソースコード片を用いて学習を行う. また, 不均一データを扱う実験ではあるが, データの比率に大きな偏りを発生させないために, 変更なしソースコード片の数を 2000 にしている. 表 4 にモデル構築に用いたデータの内訳を示す.

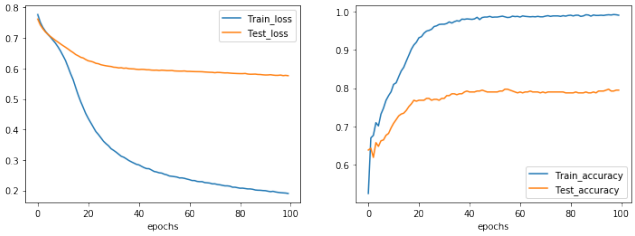


**図 1** 均一 2 分類モデルでの誤差（左）と正答率（右）

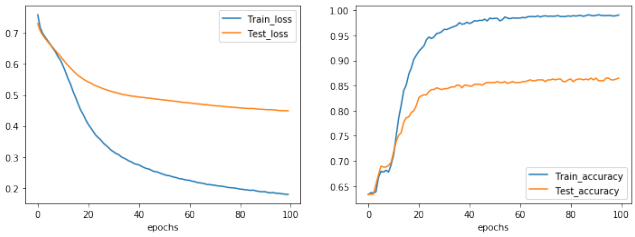


**図 2** 不均一 2 分類モデルでの誤差（左）と正答率（右）

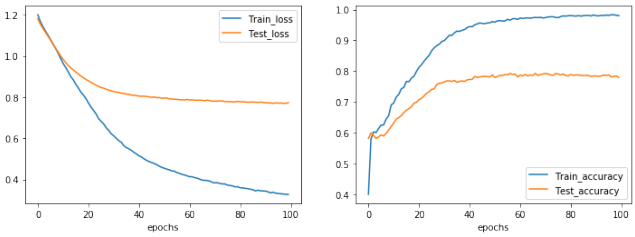


**図 3** 均一 3 分類モデルでの誤差（左）と正答率（右）

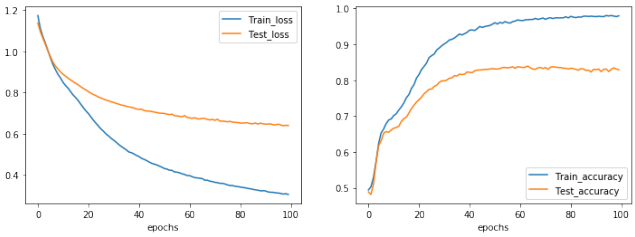


**図 4** 不均一 3 分類モデルでの誤差（左）と正答率（右）

均一モデルの学習過程における訓練データの正答率, テストデータの正答率, 訓練誤差, テスト誤差のグラフを図 3 に示す. また, 不均一モデルの学習過程における訓練データの正答率, テストデータの正答率, 訓練誤差, テスト誤差のグラフを図 4 に示す.

2 分類モデルと同様に, 学習が進むにつれ誤差は小さく, 正答率は大きくなっていることから, 業務ソフトウェアのみでの 3 分類を行う学習モデルも構築可能であると分かる. 汎化が進みにくくなる境界として 30 エポックが挙げられる. よって 4.2 節での実験ではエポック数 30 を用いて評価を行う.

均一, 不均一の違いとして，2 分類のものと同様の傾向が見られる. 次の実験によってどちらのモデルが有用的であるか判断する.

**表 5** 2 分類での均一, 不均一モデルでの交差検証による平均 AUC

| | AUC |
|---|---|
| 均一データ | 0.867 |
| 不均一データ | 0.886 |

**表 6** 2 分類での Precision，Recall

| | 修正内容 | Precision | Recall |
|---|---|---|---|
| 均一データ | バグ修正 | 0.775 | 0.738 |
| | 仕様変更 | 0.744 | 0.780 |
| 不均一データ | バグ修正 | 0.832 | 0.711 |
| | 仕様変更 | 0.854 | 0.916 |

**表 7** 3 分類での均一, 不均一モデルでの交差検証による平均 AUC

| | AUC |
|---|---|
| 均一データ | 0.897 |
| 不均一データ | 0.917 |

**表 8** 3 分類での Precision，Recall

| | 修正内容 | Precision | Recall |
|---|---|---|---|
| 均一データ | バグ修正 | 0.789 | 0.663 |
| | 仕様変更 | 0.697 | 0.707 |
| | 変更なし | 0.746 | 0.852 |
| 不均一データ | バグ修正 | 0.707 | 0.531 |
| | 仕様変更 | 0.719 | 0.790 |
| | 変更なし | 0.836 | 0.869 |

### 4.2 モデルの評価

ここでは 4.1 節で汎化が進んだエポック数を用いて 5 部交差検証を行う. 学習モデルの評価指標として ROC 曲線から得られる AUC を用いている.

**2 分類モデルの評価**

エポック数 20 で 5 部交差検証を用いて, 均一データ, 不均一データでのモデルを評価した. 表 5 に結果を示す. また, 分類ごとにおける精度の評価を行った. 評価の指標として Precision，Recall を用いる. バグ修正, 仕様変更, 変更なしの分類ごとに指標の値を算出した. 表 6, 8 に 2 分類，3 分類モデルにおける Precision，Recall の値を示す.

**3 分類モデルの評価**

エポック数 30 で 5 部交差検証を用いて, 均一データ, 不均一データでのモデルを評価した. 表 7 に結果を示す.

表 5, 7 より，2 分類，3 分類の両モデルとも高い精度で予測できていることがわかる. また均一データでの予測に比べ, 不均一データでの方が予測の精度が高いことがわかる.

表 6, 8 より, 仕様変更については均一データに比べ不均一データの方が Precision，Recall 共に値が大きくなっている. また修正なしに関しても, 不均一データでの方が指標の値が良くなっている. しかし, バグ修正に注目すると 2 分類

での Precison を除き, 不均一データの方が指標の値が悪くなっている.

2 分類にて, 仕様変更の指標が不均一データにて両方とも良くなったのは, 学習に用いた仕様変更のデータの増加が起因していると考えられる. しかし, 仕様変更に比べバグ修正のデータ数が少なくなったためバグ修正での Recall の値が小さくなり, 精度は向上しているが予測から漏れてしまうデータが増えてしまっていることが分かる.

3 分類では，2 分類同様に不均一データでの方が仕様変更の指標が向上しており, 変更なしについても同様の変化が見られる. しかし, バグ修正での指標の値が均一データに比べ, 不均一での値が大きく減少していることが分かる. 特に，Recall に注目すると不均一データでは実際のバグ修正のうち, 約半数しか検出できていないことが読み取れる．3 分類では 2 分類以上に不均一データにおけるバグ修正の指標の低下が見られた．3 分類での AUC の値は不均一データでの方が均一データより向上しているが, これはバグ修正の予測精度の低下による影響より, 仕様変更, 修正なしでの予想精度の向上の影響が大きいことが読み取れる.

学習に用いるデータの数が増えると予測精度が向上するが, すべての分類の精度が向上するわけではなく, データが少ない分類の精度は低下する傾向がある. 特に 3 分類モデルではそのような傾向が強く見られた. 不具合予測において, バグ発生箇所に関する予測精度は重要であるので, バグ修正のデータ数の割合が小さくならないようにモデルを構築することが必要と考える. 次章では均一 3 分類モデルを用いて考察を行う.

## 5. 考察

### 5.1 Grad-CAM 法による判断根拠の可視化

ここでは,Grad-CAM 法 [3] を用いて実験において W-CNN が分類を行うとき, どのような単語に注目していたのかを数値化し, 重要と判断された単語を中心にそれぞれの分類での傾向を考察する.

Grad-CAM 法は，CNN を用いた画像分類を行った際に CNN が着目していた箇所をヒートマップで可視化する手法である. 本研究では入力がテキストに対してもヒートマップを作成し, 単語の重要度としてヒートマップの画素値を出力できるように変更を加えた. 単語の重要度として，CNN の最後の層の勾配を利用している. 実装した W-CNN ではフィルタの高さが 3，4，5 の 3 種類の畳み込み層が並行に処理を行っているため, それぞれの畳み込み層での勾配のうちの最大値を重要度としている.

本章で重要度の出力に利用するモデルとして, バグ修正, 仕様変更, 修正なしのそれぞれの分類の特徴を分析したいので, 均一データによる 3 分類モデルを用いる．重要度の出力対象としてテストデータで予測の結果が実際の分類と合致していたソースコード片のみとする. 表 9 に考察に用いた分類ごとの正解データの数を示す. また, 図 5 はソースコード片の重要度をグラフ化した例を示している．0 埋めされた部分の単語は“<PAD/>”で表されている.

**表 9** それぞれの分類における正解データ数

| | 正解データ数 | 総データ数 |
|---|---|---|
| バグ修正 | 149 | 196 |
| 仕様変更 | 151 | 223 |
| 変更なし | 164 | 204 |

単語の傾向を調査する方法として, それぞれの分類別にテストデータのソースコード片に現れた単語の重要度, 出現回数, 単語ごとの平均重要度を算出したデータを作成した. データからソースコード片中に複数回以上出現し, 平均重要度が高い単語をランダムに抽出し業務ソフトウェア中での使われ方等を調べた.

**バグ修正**

バグ修正のソースコード片からは, 業務ソフトウェア固有の変数名が重要度の高い単語として多く見られた. 特に, データベースに書き込み, 読み出しを行う関数や SQL で処理を行うための文字列を保持する変数といったものが見られた. このことから, データベース関連処理を行っている周辺箇所にバグが多く発生していると予想できる. また，1 つのソースコード片中に複数回現れている場合が多かった. 高い重要度を持つ業務プロジェクト特有の変数名について, 他の分類のソースコード片中における重要度を調査したところ, 多くの場合でバグ修正のみで高い重要度を持っていた. 特に, 出現回数は多くないがバグ修正のソースコード片のみにしか出現しない単語が多く存在した. こうしたことから単語自体がバグ修正に大きく関係している可能性があり, 修正を行う箇所の目安として用いることができるのではないかと考えられる.

また, 変数名だけでなく制御文等のキーワードについても調査した. キーワードの重要度はソースコード片ごとに大きく変わってくる. これはキーワードの前後の単語によってキーワード自体の重要度が変わってくると推測する.

**仕様変更**

仕様変更のソースコード片からは, 他 2 つの分類に比べ 0 埋め部分の重要度が高い, ソースコード中の単語では先頭の単語 (クラス名) の重要度が高い傾向が見られた．また, バグ修正と同様にソースコード片中に複数回現れている単語, 特に変数が多い.

0 埋めについて, バグ修正, 修正なしでの重要度は 0 に近い値のものが多かったが, 仕様変更のみで重要度が高くなっていることがわかった. この特徴が仕様変更に分類する要因の一つになっているが，0 埋め部分の重要度が高くなる原因は特定できなかったので, 原因究明の検討が必要である.

クラス名の重要度が大きいことから, クラスの分類で仕

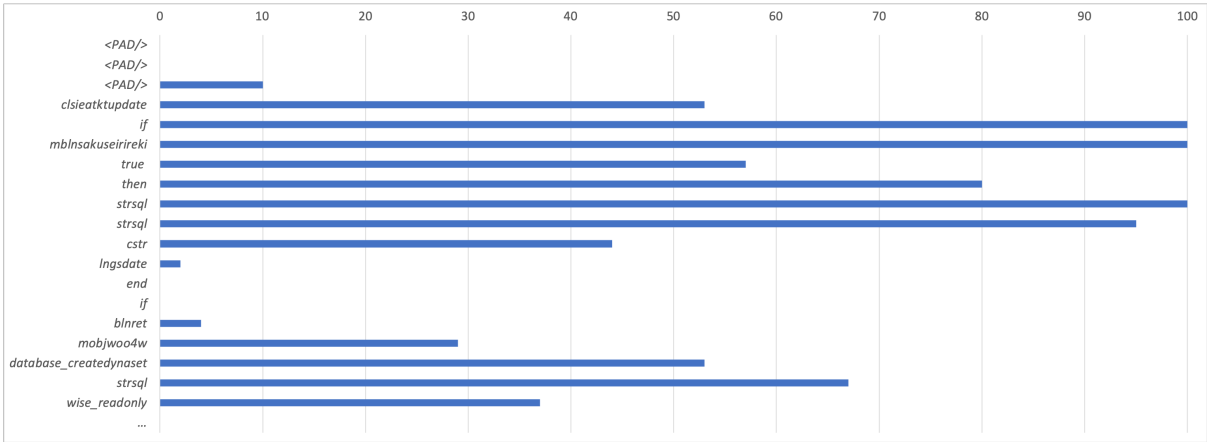


**図 5** ソースコード片の重要度のグラフ例

様変更を判別できると考えたが, 他の分類での重要度を調べたところ分類間で大きな差があるクラス名は少数であり, 大半のものはどの分類でもある程度の重要度を持っていることからクラス名のみでの分類は難しいと考える. 次に, クラス名以外で重要度の高い単語が業務ソフトウェア中でどのように使用されているか調査したところ，if 文等の分岐におけるフラグの役割を持つ変数, 単なるループカウンタなどが多く, バグ修正とは異なる特徴が見られた. 単語の傾向としては処理の流れに関連する情報を持つものと考えられる. また, ソースコード片として変数宣言部が散見された. そのようなソースコード片では型に関するキーワードの重要度が大きい傾向が見られた.

**修正なし**

修正なしのソースコード片中の単語の重要度を調べたところ, 他の分類に比べて VisualStdio の windows フォームデザイナによって生成される変数, 関数名など, 業務ソフトウェア固有のものではない単語やグローバル変数, 関数などが多く見られた. ただし, 修正なしのソースコード片は修正履歴が存在しない部分を収集したもので, 今後も修正されないという保証はない. また, 修正なしで見られた単語, 特にグローバル変数, 関数は広い範囲で使われているため, 修正なしのソースコード片として収集されやすかったのではないかと考える.

## 6. おわりに

### 6.1 まとめ

本研究では，VB.net で書かれた業務ソフトウェアを対象として近藤らの W-CNN を参考に構築した学習モデルを用いて不具合予測を行った．W-CNN は変更ソースコード片を用いる必要があり, 対象の業務ソフトウェアから得られる変更履歴情報は学習モデルを構築するにはかなり少ないと思われたが, 構築は可能であるということが示せた.

また, バグ修正, 仕様変更の 2 分類での予測だけでなくバグ修正, 仕様変更, 修正履歴なしの 3 分類での予測も可能であることが分かった. さらに, それぞれの分類のデータ数が不均一であっても, 均一であるときに比べ精度が高く出ることが分かったが, 分類ごとの精度に注目するとデータが少ない分類, 今回ではバグ修正の予測精度が低下していた. よってデータの総数が少ない場合でも, 分類ごとのデータ数を均一にしてモデル構築をした方が良いと考えられる.

さらに，Grad-CAM 法を利用することで分類する際に重要視されている単語を抽出した. 重要度が高い単語を中心に調べたところ, 分類ごとに違った特徴を持つ単語を見つけることができた.

### 6.2 今後の課題

今後の課題として以下が挙げられる.

- 対象としている業務ソフトウェアは機能ごとに幾つかのソリューションに分かれている. 今回, 予測を行う対象としてソースコード片を抽出したのは 1 つのソリューションのみである. 複数のソリューションを対象に行い, 今回の場合と精度がどのように変化するかを調査し, より広い範囲で不具合予測を行えるように前処理, ネットワークの改善を行う必要がある.
- ソースコード片をソフトウェア中のコメントに基づいて自動的に抽出を行っているが, 修正の時系列を考慮せずに抽出を行っているため, 当時のソフトウェアに含まれないソースコード片を抽出している可能性がある. ソースコード片の妥当性を見直し, ソースコード片抽出ツールの改善を行う必要がある.
- 抽出したソースコード片に対してどの分類に属するかのラベルづけを行っているが, 今回この作業を人手によって行っている. 修正履歴情報に ID ごとに修正内容が自然言語で記載されており, この内容によってラベルづけを行った. したがって誤りが混入していたり, 複数人で行った場合個人差が発生する恐れがある. ラベルの割り振りに関して, 自然言語処理を用いて自動的に行うなど高精度かつ定量的な判断が求められる.

- 本研究で実装した学習モデルでは入力としてソースコード片を用いており, 不具合予測を行うソースコードを自身で定める必要がある. 今後の不具合予測として, ソースコードファイル自体などを入力として, 分類だけでなく不具合が発生するであろう箇所を予測するモデルの開発が求められる.
- 考察で重要度の高い単語を抽出しソフトウェア中での役割を調べたが, 調査は人手によって行い明確な判断基準を設けて特徴の傾向を調べたわけではない. 特にソフトウェア中での役割については不明な点が多く, 実際にこの業務ソフトウェア作成に携わった方々に確認してもらう必要性がある.

**謝辞** 本研究を進めるにあたり，業務ソフトウェアのソースコードおよび欠陥修正情報を提供いただいた (株) 東日本技術研究所に感謝致します.

## 参考文献

[1] 近藤将成， 森啓太， 水野修， 崔銀恵ほか：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，情報処理学会論文誌，Vol. 59, No. 4, pp. 1250–1261 (2018).
[2] 川田秀司，安田康二， 山下剛，山元和子ほか：深層学習によるプロジェクトを跨いだソフトウェア不具合混入予測，ソフトウェアエンジニアリングシンポジウム 2019 論文集， Vol. 2019, pp. 237–244 (2019).
[3] Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D. and Batra, D.: Grad-cam: Visual explanations from deep networks via gradient-based localization, *Proceedings of the IEEE international conference on computer vision*, pp. 618–626 (2017).
[4] Kim, Y.: Convolutional neural networks for sentence classification, *arXiv preprint arXiv:1408.5882* (2014).
[5] Yang, X., Lo, D., Xia, X., Zhang, Y. and Sun, J.: Deep learning for just-in-time defect prediction, *2015 IEEE International Conference on Software Quality, Reliability and Security*, IEEE, pp. 17–26 (2015).
[6] : Roslyn, https://github.com/dotnet/roslyn.
[7] 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 preprint arXiv:1603.04467* (2016).
[8] Chollet, F. et al.: Keras, https://keras.io (2015).