ソフトウェア不具合予測への画像分類手法の適用

廣瀬　早都希
京都工芸繊維大学
大学院工芸科学研究科 情報工学専攻
s-hirose@se.is.kit.ac.jp

水野 修
京都工芸繊維大学
情報工学・人間科学系
o-mizuno@kit.ac.jp

要旨

ソフトウェアの品質を確保することはソフトウェア開発において重要である．そして，ソフトウェアの品質を保証するためには，ソフトウェアに含まれる不具合を早期に発見する必要がある．その課題に対し，本研究ではソフトウェアの不具合予測を行うシステムの試作を行なった．本研究では，ソースコードの変更点のテキストデータを画像化し，機械学習の *1* つであるニューラルネットワークを用いて画像分類を行うことで，ソースコードに不具合がふくまれるか否かを分類する手法を提案する．その手法において，全結合ニューラルネットワークと畳み込みニューラルネットワークの *2* 種類のモデルを作成し，学習・分類を行なった．その結果，全結合ニューラルネットワークでは，学習が正しく行われなかった．また，畳み込みニューラルネットワークでは，学習は行われたものの過学習となってしまったため，汎化性能がなく，画像分類ができなかった．今後，ネットワークの構成方法などを変更することで，モデルの改良を目指していく．

1　はじめに

現在，ソフトウェアは技術的及び経済的な発展において重要な役割を持っている．そのため，ソフトウェアには高い品質が求められる．しかしながら，ソフトウェア開発において高品質を保つことは，人的・時間的理由から難しい問題とされている．また，ソフトウェア開発過程における大部分を人間の手によって，直接的または間接的に行われるため，人的エラーを完全に排除することは難しく，ソフトウェアの品質保証においての問題の 1 つとされている．

こういった問題を解決する方法の 1 つとして，ソフトウェア不具合予測という手法がある．ソフトウェア不具合予測とは，過去のソフトウェア開発情報を利用することで，現在開発途中のソフトウェアに潜む不具合を予測することである．これにより，ソフトウェアに含まれる不具合を早期に発見できるため，後に重大な不具合を生じさせるリスクを減らすことができる．また，最終的にテストやメンテナンスにかけるコストを減らすことに繋がる．

ソフトウェア不具合予測に関する研究は古くから行われてきているが，Commit Guru[1] [1] というリポジトリの変更点 (commit) を解析してデータを与えるソフトウェア不具合予測用のツールが公開されたことからより進んできている分野である．近年では，ソフトウェア不具合予測に機械学習を用いた研究が Yang ら [2]，Wang ら [3]，Sharma ら [4] によって行われている．これらの研究では，ソフトウェア不具合予測に用いる特徴セットをより良いものとするために機械学習を用いて良い特徴を生成する手法が取られていた．つまり，機械学習の学習器部分は利用されているが，分類器としては重きを置いていない研究である．一方，機械学習をソフトウェア不具合予測の分類器として利用する手法が近藤ら [5] によって提案されている．この研究では，畳み込みニューラルネットワーク (CNN:Convolutional Neural Networks) を用いて，変更 (追加・修正) されたソースコードのテキスト情報から，変更後のソースコードに不具合が含まれているか否かを分類するという手法が取られている．この研究ではソフトウェアに潜む不具合を高精度で検出することができていた．

[1] Commit Guru:http://commit.guru

そこで，本研究では，近藤ら [5] の研究と同様に，機械学習を分類機として利用する．ただし，ソースコードの変更点における追加・修正情報をテキストとしてではなく，画像情報として取り出しソフトウェア不具合予測を行う手法を提案する．今回，テキスト情報を用いるのではなく，画像情報を取り扱うことにしたのは，人の目によってソースコードに含まれる不具合を発見できるのであれば，画像分類により不具合有無が判別できるのではないかと考えたからである．先行研究 [6] にてソースコードのインデントと複雑さには相関があるという結果が得られている．ソースコードの複雑さは不具合有無に関係しているため，ソースコードのインデントという画像で把握できる情報と複雑さに相関があるならば，画像分類によっても不具合を発見できるのではないかと考えた．

先行研究にて，ソースコードの変更点でのソフトウェア不具合予測モデルが有意であることが示されている [7–10]．変更点でのソフトウェア不具合予測では，変更が行われた段階で，その変更に対する不具合予測が行われるため，開発者に対するフィードバックが早く，また，開発者は不具合が検出された箇所のみレビューを行えば良いので，非常に簡単に不具合を解消することができる．これより，本研究でもソースコードの変更点でのソフトウェア不具合予測を行うこととする．

2 研究設問

本研究の目的はソフトウェア不具合予測をより簡易に行えるようにする方法を提案することである．先行研究 [5] ではテキスト情報に対して機械学習を行い，ソフトウェア不具合予測をすることに関して高い成果をおさめている．本研究ではソースコードの画像としての情報に着目し，ソースコードの変更点情報の画像を用いて機械学習を行い，変更点における不具合の有無を判定する手法を提案する．

よって，本研究における研究設問 (RQ:Research Question) は以下のように設定する．

- RQ1: ソースコードの変更点情報の画像で機械学習による学習は可能か?
- RQ2: 機械学習による画像分類でソフトウェア不具合検出を行った結果，精度はどの程度か?

3 準備

3.1 Commit Guru

Commit Guru [1] は Emad Shihab らによって公開されている，リポジトリを解析することで得られるデータをまとめて提供する Web サイトである．Commit Guru では，解析するリポジトリの各変更点に関して 13 項目の指標を記録する．ここで使用される 13 項目の指標は先行研究 [10] で設定されているものと同様に以下のものである．

- Number of Subsystems(NS:変更されたサブシステムの数)
- Number of Directories(ND:変更されたディレクトリの数)
- Number of Files(NF:変更されたファイルの数)
- Entropy(各ファイルにおける変更されたコードの分布)
- Line Added(LA: 追加されたコード行)
- Line Deleted(LD:削除されたコード行)
- Lines Total(LT:変更前ファイル内のコード行),
- Number Developers(NDEV:変更ファイルを編集した開発者の数)
- Age of changes (AGE:最後の変更からの時間)
- Number of unique changes(NUC:変更ファイルに含まれる固有の変更の数)
- Developer Experience(EXP:開発者の経験数 (総 commit 数))
- Recent Experience(REXP:開発者の最近の経験数 (日付で重み付けされた commit 数))
- Experience on a subsystem(SEXP:サブシステム上での開発者の経験数 (総 commit 数))

これら 13 の指標に加えて，変更点での不具合予測結果をラベル付けして，まとめたデータを公開している．Commit Guru で行われる不具合予測は，ログに含まれる commit メッセージを利用したものである．先行研

究 [11] に基づいたキーワード，’bug’，’fix’，’wrong’，’error’，’fail’，’problem’，’patch’ が commit メッセージに含まれていた場合，その変更点では不具合が修正されたと考えられる．不具合を修正している変更点が発見されると，そこから不具合が発生した変更点を特定する．そして，不具合が含まれている変更点の場合 bug=1 とし，不具合が含まれていない変更点の場合 bug=0 としてラベル付けを行う．ソフトウェア不具合予測モデルの研究において重要であるとされてきた実験データの公開性と透明性を保つため，本研究では Commit Guru で得られるデータを用いることとする．

3.2 Keras

Keras[2]は Python(汎用のプログラミング言語) で書かれており，迅速な実験を可能とするために開発された，TensorFlow や CNTK，Theano 上で実行可能な高水準のニューラルネットワークライブラリである．次のような場合に，Keras を利用すると良いとされている．

- ユーザーの技能やモジュールの内容，また拡張性によるが，簡単で，かつ短時間で試作の作成を行う場合
- 畳み込みニューラルネットワークや再帰型ニューラルネットワーク (RNN: Reccurent Neural Networks) を用いたり，これらの組み合わせでニューラルネットワークを作成を行う場合
- CPU 上，GPU 上で操作感を変えずに動作を行う場合

本研究では提案手法の試作が主目的となるため，短期間で開発が可能な Keras を利用した．

3.3 TensorFlow

TensorFlow[3]とは，Google が開発し公開している，機械学習で使用するためのソフトウェアライブラリである．TensorFlow では，ニューラルネットワークを柔軟に構築することができる．例えば，ニューラルネットワークのモデルをツールで組み立てたり，Python で直感的に書いたりなど，ニューラルネットワークを柔軟に記述できる．また，機械学習は計算量が多くなるため，グラフィックボードの計算能力を使用した演算が一般的であるが，CPU 演算や GPU 演算といった使用できるコードが異なるフレームワークがある．異なるフレームワークを使用するたびにコードを書き換えるのは手間がかかる．しかし，TensorFlow を用いるとコードを書き換える手間がなく，CPU と GPU 共に同じコードを使うことができるため，これらの切り替えがスムーズに行える．

本研究ではニューラルネットワークモデルを作成する際に 3.2 節で述べた Keras と TensorFlow を使用することとする．

[2]Keras:https://keras.io/ja/
[3]TensorFlow:https://www.tensorflow.org

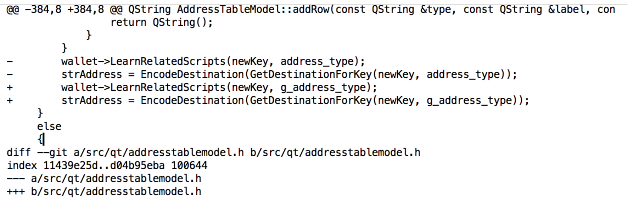
4 実験準備

4.1 使用するソースコード

本研究では，Commit Guru で得られるデータを利用するため，Commit Guru で公開されている Git リポジトリの中から，bitcoin という仮想通貨を取り扱うシステムのプロジェクトを取得し，このソースコードを使用して，実験を行った．bitcoin のソースコードは汎用プログラミング言語の 1 つである C++を用いて書かれている．実験に使用するプロジェクトに bitcoin を選択したのは，十分な commit があることと，汎用プログラミング言語で書かれていることである．このプロジェクトの Git 上に記録されている commit 数は約 16000 個だが，今回は各コミットに対する不具合の有無を Commit Guru によるラベル付けから得ているため，Commit Guru によりすでに不具合ラベル付けがなされていた約 15000 個の commit を対象とする．

4.2 ソースコードの変更点の画像化

本研究では，画像分類による不具合予測を行うため，入力データとして使用する画像を用意する必要がある．取得したソースコードから変更点のみの画像化を行うため，今回は `git diff` を使用し，commit 同士の差分をとることとする．git diff による出力例を図 1 に示す．’-’ から始まる行が commit で削除された箇所で，’+’ から始まる行が commit で追加・修正された箇所である．今回は追加・修正された箇所に焦点を当てることとする．git diff で取り出した差分から ’+’ から始まる行を取り出し，取り出した行から先頭の‘ + ’を削除する．こうして取り出されたテキストを，画像サイズ 1024×1024 で RGB 画像として書き込む．なお，フォントは “NotoMono-Regular” と



```
@@ -384,8 +384,8 @@ QString AddressTableModel::addRow(const QString &type, const QString &label, con
                 return QString();
             }
         }
-        wallet->LearnRelatedScripts(newKey, address_type);
-        strAddress = EncodeDestination(GetDestinationForKey(newKey, address_type));
+        wallet->LearnRelatedScripts(newKey, g_address_type);
+        strAddress = EncodeDestination(GetDestinationForKey(newKey, g_address_type));
     }
     else
     {
diff --git a/src/qt/addresstablemodel.h b/src/qt/addresstablemodel.h
index 11439e25d..d04b95eba 100644
--- a/src/qt/addresstablemodel.h
+++ b/src/qt/addresstablemodel.h
```

図 1. git diff コマンドの出力例

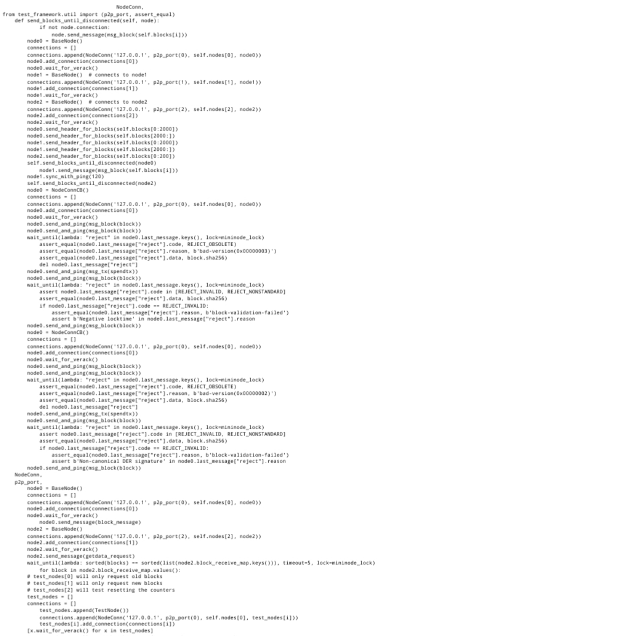


図 2. 変更点の画像化例

いう Google が公開している等幅フォントを使用し，フォントサイズは 8 ポイントとした．作成した画像例を図 2 に示す．画像例から分かるように，今回の実験で用いた画像は白黒画像であるが，RGB 画像として扱っている．これは今後の研究ではシンタックスハイライトなどの色情報を取り入れることを考えているためである．なお，画像サイズを超えるだけのテキストが抽出された場合，超えた分のテキストは縦横共に切り捨てた．機械学習を行う際は，画像サイズをこのまま用いると大きいため，計算量が大きくなりすぎる．よって，計算量を減らすために画像サイズを 256 × 256 に変換して利用する．

4.3 Commit Guru から不具合ラベルを取得

Commit Guru で公開されているデータを本研究では使用する．特に使用するデータは変更点における不具合の有無である．ソースコードの変更点それぞれで得られた変更情報のうち，追加・修正部分のみを取り出したテキスト画像に対して，Commit Guru から取得した各変更点の不具合の有無をラベルとして設定する．これにより，ソースコードの各変更情報画像に対して，不具合の有無情報を付与できる．

4.4 使用した機械学習モデルの構造

本研究では，機械学習モデルとして 2 種類のネットワークを利用して，その性能を比較した．ここでは，実験で使用した機械学習モデルの構造について説明する．

4.4.1 全結合ニューラルネットワーク

ニューラルネットワークとは，機械学習の手法の 1 つであり，人間の脳内にある神経細胞 (ニューロン) とその繋がりである神経回路網を人工ニューロンという数式的なモデルで表現したものである．入力層，中間層，出力層から構成され，層と層の間にニューロン同士の繋がりの強さを表す重み w が存在する．一つ前の階層から次の層へと入力が行われる時，入力される値に重み w をかけていき，最終的な値を出力する．学習時は，出力されたデータと教師データの誤差を計算し，出力データが教師データに近づくように重み w の調整を行う．分類時は，学習時に得られた重み w を利用して結果を出力する．全結合ニューラルネットワークモデルの概要は図 3 のようになっている．全結合ニューラルネットワークでは，1 つ前の階層から次の階層へとユニットの更新が行われる時に前の階層の全てのユニットにそれぞれの重み w をかけ，それらの総和をとる．この総和に対して活性化関数という，入力信号の総和を出力信号に変換する関数を用いて，次の階層のユニットを決定する．しかし，全てのユニットを使用して，次の階層のユニットを設定すると，過学習 (overfitting) が生じる危険性がある．過学習とは学習データに対しては学習されているが，未知のデータ (テストデータ) に対しては適合できていない状態となることである．つまり，過学習が行われたニューラルネットワークでは汎化性能がないため，未知のデータでの分類ができない．そこで，過学習防止のために Dropout という手法を取る．Dropout では，ニューラルネットワークでの学習の際に，次の階層への更新において，前階層の中のユニット全てを用いて次階層のユニットを求めるのではなく，前階層のユニットのうちのいくつかを無効

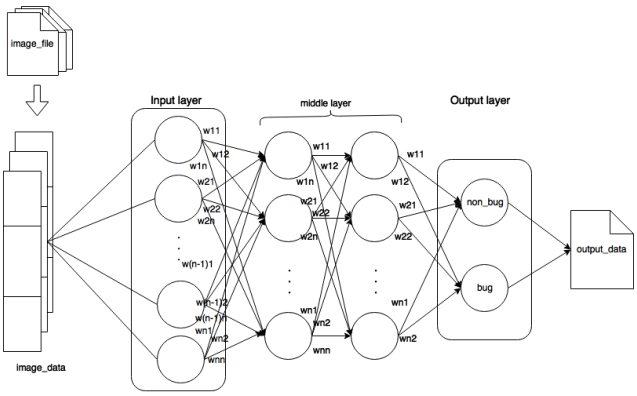


図 3. 全結合ニューラルネットワークモデルの概要

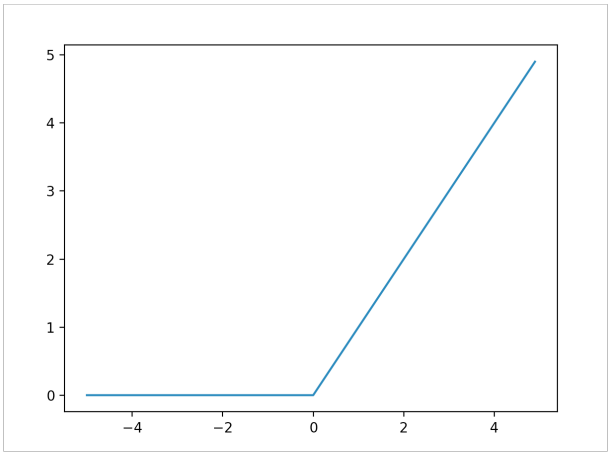


図 4. ReLU 関数

にして (存在しないものとして扱い) 学習を行う．そして，次の更新の際はまた別のユニットを無効にして学習を行うことを繰り返す．これにより，学習に際してネットワークの自由度を強制的に上げることで汎化性能を上げて過学習を避けることができる．

全結合ニューラルネットワークを使用するため，4.2 節で作成した画像情報を一次元配列に格納した．実験に使用する画像は RGB の 3 チャネルで作成しているので，一次元配列に最初の 1/3 が Red，次の 1/3 が Green，最後の 1/3 が Blue と並ぶように格納した．本モデルでは中間層は 2 層とし，各中間層のユニット数は 200 個で 20%のユニットが Dropout となるように作成した．また，活性化関数には以下の ReLU 関数 (式 1 図 4) を使用する．出力層では不具合の有無の 2 値を最終的な出力となるようにし，活性化関数として softmax 関数 (式 2 図 5) を用いる．

$$Relu(x) = \begin{cases} x & (x \leq 0) \\ 0 & (x < 0)ring \end{cases} \tag{1}$$

$$softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \quad (i = 1, 2, \cdots, n) \tag{2}$$

4.4.2 畳み込みニューラルネットワーク

畳み込みニューラルネットワークは画像分類においてより高い成果をあげている機械学習アルゴリズムである．このニューラルネットワークは「畳み込み層」「プーリング層」「全結合層」を積み上げることで構成される．図 6 に示すのは畳み込みニューラルネットワークモデルの概要である．

畳み込み層では，入力された画像の特徴を捉えるためのフィルタを用いて，入力画像の特徴を捉えた画像データの様な二次元配列を作成し，この配列の各値に対して活性化関数を適用させた値を出力をとする．学習を行う際は，このフィルタの値がどの様になるかの学習を行い，分類を行う際は，学習によって作成されたフィルタを用いて，入力画像の特徴を抽出する．プーリング層では，畳み込み層で出力された二次元配列の中から有効な値だけを残す処理を行う．つまり，二次元配列を区切り，区切った小領域から有効な値を選択する．これにより画像分類を行う上であまり重要でない情報を削り，より特徴的な情報のみを残すことができる．全結合層は 4.4.1 節で説明した通りである．全結合層では一次元配列を取り扱うため，全結合層に入る前に二次元配列を平滑化し，一次元配列とする必要がある．

今回作成した畳み込みニューラルネットワークでは，まず入力層から畳み込み層を 2 個積み上げた後，プーリング層を積み上げた．なおプーリング層ではマックスプーリングという手法を用いる．マックスプーリングとは，それぞれの小領域に対して最大の値を選択するものである．この畳み込み層では，どちらもフィルタ数は 32 個とし，サイズは 3×3 とする．プーリング層では，マックスプーリングを適用する領域サイズを 2×2 とする．その後，さらに，畳み込み層とプーリング層を 1 個ずつ積

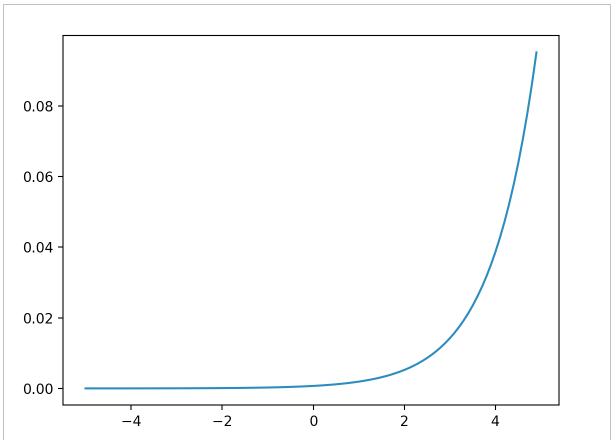


図 5. softmax 関数

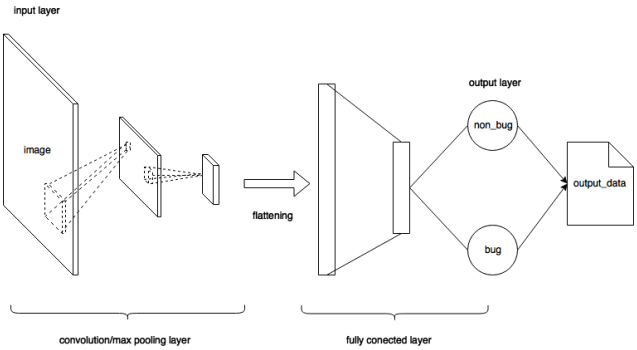


図 6. 畳み込みニューラルネットワークモデルの概要

み上げた後，得られたデータを平滑化し，全結合層で出力層と繋ぐこととする．こちらの畳み込み層では，フィルタ数を 64 個とし，サイズは 3×3 とする．プーリング層は先ほどと同じくマックスプーリングを適用し，領域サイズを 2×2 とする．畳み込み層と全結合層で利用する活性化関数は ReLU 関数とする．また，全結合層ではユニット数を 128 個とし，50%を Dropout となるようにした．全結合型ニューラルネットワークと同じく，最終的な出力は不具合の有無の 2 値とし，活性化関数として softmax 関数を用いる．

表 1. 混合行列

		分類結果	
		bug	non-bug
真のクラス	bug	TP	FN
	non-bug	FP	TN

4.5 評価尺度

本研究における評価尺度は，混合行列 (Confusion Matrix) と呼ばれる行列から求められる値に従って，定量的に評価を行うこととする．混合行列は表 1 で示す．混合行列とは，分類処理を行った際に，分類結果の値 (本研究では不具合が有無の 2 値) とその正誤 (真か偽か) についてまとめた表である．表で示されている各項目は以下のように示される．

- 真陽性 (TP:True Positive): 分類結果が不具合有り (bug) で，予測結果も不具合有り (bug) の場合
- 偽陽性 (FP:False Positive): 分類結果が不具合有り (bug) で，予測結果が不具合無し (non-bug) の場合
- 偽陰性 (FN:False Negative): 分類結果が不具合無し (non-bug) で，予測結果が不具合有り (bug) の場合
- 真陰性 (TN:True Negative): 分類結果が不具合無し (non-bug) で，予測結果も不具合無し (non-bug) の場合

これらの値を用いて，本研究で用いる指標，Recall(再現率)，Precision(適合率)，F1-score(F1 値)，Accuracy(正解率) の値を求める. 各指標について次に述べる.

4.5.1 Recall(再現率)

Recall(再現率) とは，実際に正であるもののうち，正であると予測されたものの割合である．つまり，実際に不具合 (bug) であったものの中から，不具合と分類されたものの割合である．混合行列 (表 1) を用いて定義すると以下のようになる．

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

4.5.2 Precision(適合率)

Precision(適合率) とは，正と予測したデータのうち，実際に正であるものの割合である．つまり，不具合と分類された結果の中から実際に不具合であったものの割合である．混合行列 (表 1) を用いて定義すると以下のようになる．

$$Precision = \frac{TP}{TP + FP} \tag{4}$$

4.5.3 F1-score(F1 値)

F1-score(F1 値) とは，recall(再現率) と Precision(適合率) 率の調和平均である．以下のように定義される．

$$F1 = \frac{2 \times Rcall \times Precision}{Recall + Precision} \tag{5}$$

4.5.4 accuracy(正解率)

accuracy(正解率) とは，分類結果全体と、真のクラスが一致している割合である．混合行列 (表 1) を用いて定義すると以下のようになる．

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{6}$$

5 結果と考察

ここでは，研究設問 (RQ) に対する結果と考察を行う．

5.1 RQ1: ソースコードの変更点情報の画像で機械学習による学習は可能か?

5.1.1 概要

ソフトウェアの品質を保つ方法として，ソフトウェアに潜む不具合を早期に発見することがある．そこで，より手軽にソフトウェア不具合予測が行える様にするために，ソースコードの変更点の画像情報から，ソースコードに潜む不具合を検出できるかを提案した．そこで機械学習を利用して学習・分類を行うことができないか実験を行った．本研究では，ソースコードの変更点情報の画像で機械学習の分類を行うにあたり，学習時間が短い全結合ニューラルネットワークと画像分類に適しているが学習時間がかかる畳み込みニューラルネットワークを作成し，学習を試みた．

表 2. 学習データ・テストデータに含まれる画像データの内訳

	nunber of files
train-bug	2,071
train-nonbug	11,984
test-bug	208
test-nonbug	1,354

今回の実験では，学習データ約 14000 個，テストデータ約 1500 個を用いて実験を行うこととするここで用意した学習データ・テストデータはともに不具合有・不具合無の両方の画像が含まれているデータである．内訳は表 2 の様になっている．

このデータを用いて，本研究で用いた全結合ニューラルネットと畳み込みニューラルネットの 2 種類のモデルについて，それぞれ学習性能を示す訓練誤差 (train loss) を記録することで，これらのモデルで学習が行われているか，検証を行う．

5.1.2 結果と考察

全結合ニューラルネットワークと畳み込みニューラルネットワークそれぞれでの学習時に記録した訓練誤差 (train loss) をグラフ化したものを図 7 と図 8 に示す. 図 7 では訓練誤差の減少が見られない．よって，全結合ニューラルネットワークでは学習が行われていないことがわかる．一方，図 8 を見るとこれらを見る訓練誤差は減少している．つまり畳み込みニューラルネットワークによる学習は可能であると考えられる．

5.2 RQ2: 機械学習による画像分類でソフトウェア不具合検出を行った結果，精度はどの程度か?

5.2.1 概要

RQ1 で畳み込みニューラルネットワークによる学習が可能であることが分かった．よって，学習可能であった畳み込みニューラルネットワークを用いて，画像分類

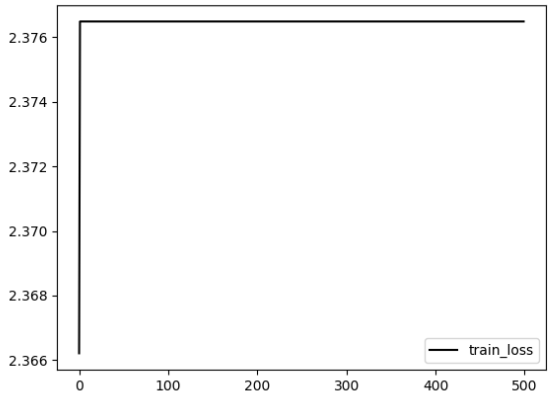


図 7. 全結合ニューラルネットワークの学習時の訓練誤差 (train loss)

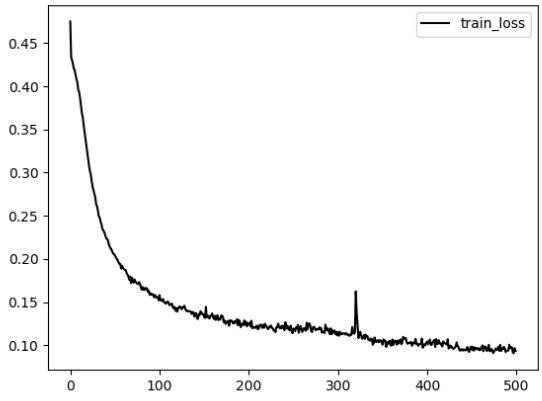


図 8. 畳み込みニューラルネットワークの学習時の訓練誤差 (train loss)

によるソフトウェア不具合検出を行い，その精度を検証する．

RQ1 で使用したデータと同様のものを用いて，畳み込みニューラルネットワークでの画像分類を行い，分類結果を記録した．さらに分類結果から，それぞれのニューラルネットワークの精度を考察した．

5.2.2 結果と考察

ソースコードの変更点の画像を用いて，畳み込みニューラルネットワークで学習を行い，分類結果を記録した．この分類結果を受け，本研究で使用した畳み込みニューラルネットワークモデルの精度について考察を行う．記録した分類結果から計算して得られた評価尺度の値を表 3 に示す. 表 3 によると，Accuracy(正解率) は高いが，Recall(再現率)，Precision(適合率)，F1-score(F1 値) はかなり低い結果となっている．使用した畳み込みニューラルネットワークの分類結果の値と真の値とで判断される混合行列 (各項目には当てはまる個数が示される) は表 4 の通りである．本研究の目的としては，ソフトウェアの変更点に潜む不具合を検出することにある．しかしながら，この結果を見ると，全ての不具合 (208 個) のうち，不具合として検出されたのはわずか 18 個である．これでは，Accuracy(正解率) が高くともソフトウェアの不具合検出器としての精度が高いとは言えない．

では，何故畳み込みニューラルネットワークを分類器として用いた時，精度が低くなったのかの考察を行う．RQ1 より，畳み込みニューラルネットワークでの学習は行われていた．しかし，この学習が正しく行われているかは不明である．そこで，畳み込みニューラルネットについて，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で，学習が正しく行われているかを確認を行った．図 9 に記録した訓練誤差 (train loss) とテスト誤差 (test loss) のグラフを示す．

これを見ると，訓練誤差 (train loss) は減少しているが，テスト誤差 (test loss) は学習回数が増す毎に上昇していることがわかる．これは，畳み込みニューラルネットワークが過学習していることを示している．つまり，今回学習した畳み込みニューラルネットワークには汎化性能は全くないと言える．

過学習となった原因を考えると，次の 2 点が挙げられる．

- 学習データ数が少ない．
- 中間層のパラメータ数が多すぎる．

前者に関しては，本研究にあたり Commit Guru で公開されているソースコードから commit 毎の変更情報を画像化するということで，データ数を増やすのが困難であったため，少ないデータ数での実験となった．そのため，作成した画像データにノイズを加えたりなどして，学習データの水増しを行ったりする必要がある．また，明らかに不具合有りのデータが少ないので，不具合有りと不

表 3. 画像分類による不具合検出の評価

	畳み込みニューラルネットワーク
Recall	0.087
Precision	0.261
F1-score	0.130
Accuracy	0.846

表 4. 畳み込みニューラルネットワークの分類結果から得られる混合行列の値

		分類結果	
		bug	non-bug
真のクラス	bug	18	190
	non-bug	51	1303

具合無しの画像データ数を揃えることも必要である．

後者に関しては，畳み込みニューラルネットワークでは，画像サイズが 256×256 の分類をするためには中間層のパラメータを大きくする必要がある．本研究で用いた中間層のパラメータは調整を行わず，適度な値を設定していた．そのため，データ数に対する中間層のパラメータが大きいため，過学習となってしまったと考えられる．よって，データ数を増やすことで改善が見られるかもしれない．また，中間層を必要最小限まで減らしたり，Dropout を適切に設定することで改善が見られる可能性も考えられる．しかし，テスト誤差を見る限りでは，全く減少が見られないため，汎用性能は上がらない可能性もある．

6 今後の課題

本研究の結果・考察 (5.1.2 節) から，全結合ニューラルネットワークでは学習が行われず，畳み込みニューラルネットワークでは学習は行われたが，過学習となってしまったため，画像分類は上手く行えなかった．そこで，今後は畳み込みニューラルネットワークが過学習をせずに学習を終え，画像分類ができるようになることを目標とする．畳み込みニューラルネットワークが過学習する原因として，以下の問題点が与えられた．

- 実験で使用した学習データ数が少ない．

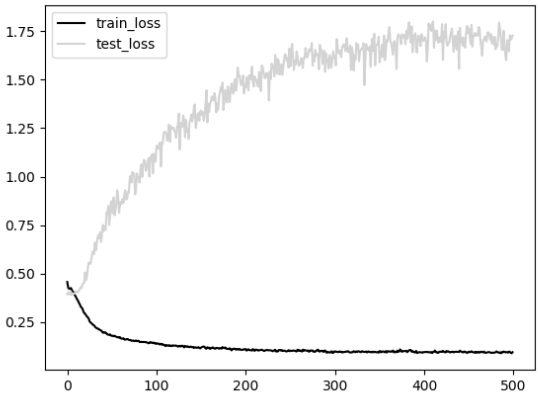


図 9. 畳み込みニューラルネットワークの訓練誤差とテスト誤差

- 実験で使用した畳み込みニューラルネットワークの中間層のパラメータ数が多すぎる．

これらの問題点を解決すれば，畳み込みニューラルネットワークでの学習が正しく行えるという仮説のもと，今後の課題を次に設定する．

1. 実験で使用するデータ数を増やす．
2. 畳み込みニューラルネットワークのモデル作成の際，適切な中間層の値を設定する．
3. 利用する画像データをシンタックスハイライトを利用したものにする．

本研究では，妥当性に対する措置を取っていなかった．今後は妥当性に対する検証も行わなければならないため，上記の他に新たに以下を今後の課題として挙げる．

1. 実験で使用するデータの偏りを減らす
2. 実験で使用するプロジェクトの種類を増やす
3. 10 重交差検証を行う
4. 作成したプログラム (畳み込みニューラルネットワーク) に不具合がないかを確認する．

また，畳み込みニューラルネットワークでは画像のどこに注目し判断しているか，画像の特徴箇所を特定することができる．よって，これからの研究で畳み込みニュー

ラルネットワークでの学習・分類ができることがわかれば，この技術を用いて人の目による不具合検出の糧としたい．

7 まとめ

本研究では，機械学習の画像分類を用いてソフトウェアに含まれている不具合を判定する研究を行った．全結合ニューラルネットワークでは実験データの特徴を捉えきれず，学習が行われなかった．そこで畳み込みニューラルネットワークを使用することにしたが，学習の際に過学習を行ってしまい，正しく画像分類は行われなかった．今後の課題としては，より多くのデータを用いて畳み込みニューラルネットワークでの学習を行うことと，畳み込みニューラルネットワークモデルにおける中間層の値を見直すことが挙げられる．これにより，畳み込みニューラルネットワークでの画像分類でソフトウェア不具合予測が行えるかを再度検証する．

参考文献

[1] C. Rosen, B. Grawi, and E. Shihab, “Commit guru: Analytics and risk prediction of software commits,” Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, pp.966–969, New York, NY, USA, July 2015.

[2] X. Yang, D. Lo, X. Xia, Y. Zhang, and J. Sun, “Deep learning for just-in-time defect prediction,” QRS’15: Proceeding of the 2015 IEEE International Conference on Software Quality, Reliability and Security, pp.17–26, Washington, DC, USA, Aug. 2015.

[3] S. Wang, T. Liu, and L. Tan, “Automatically learning semantic features for defect prediction,” ICSE ’16: Proceedings of the 38th International Conference on Software Engineering, pp.297–308, New York, NY, USA, May 2016.

[4] R. Sharma and P. Kakkar, “Software module fault prediction using convolutional neural network with feature selection,” International Journal of Software of Software Engineering and Its Applications, vol.10, no.12, pp.307–318, 2016.

[5] 近藤将成，森　啓太，水野　修，崔　銀惠，“深層学習による不具合混入コミットの予測と評価,” ソフトウェアエンジニアリングシンポジウム 2017 論文集，vol.2017，pp.35–45，Aug. 2017.

[6] A. Hindle, M. Godfrey, and R. Holt, “Reading beside the lines: Indentation as a proxy for complexity metrics,” In Program Comprehension, 2008. ICPC 2008. The 16th IEEE International Conference on, pp.133–142, May 2008.

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

[8] L. Aversano, L. Cerulo, and C.D. Grosso, “Learning from bug-introducing changes to prevent fault prone code,” IWPSE’07: Ninth international workshop on Principles of software evolution: in conjunction with the 6th ESEC/FSE joint meeting, pp.19–26, NY, USA, July 2007.

[9] T. Jiang, L. Tan, and S. Kim, “Personalized defect prediction,” ASE’13: Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering, pp.279–289, NJ, USA, Nov. 2013.

[10] 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 Transactions on Software Engineering, vol.39, no.6, pp.757–773, June 2013.

[11] A. Hindle, D.M. German, and R. Holt, “What do large commits tell us?: a taxonomical study of large commits,” MSR’08: Proceedings of the 2008 international working conference on Mining software repositories, pp.99–108, New York, NY, USA, May 2008.