CNN-BI システムの複数モデルによる精度向上のための研究

小川 一彦
放送大学
kgg03771@nifty.com

中谷 多哉子
放送大学
tinakatani@ouj.ac.jp

要旨

ソフトウェアの品質を向上させるため，これまで多くの研究が行われてきた．その方法の一つにソースコードの不具合を推論する方法がある．不具合があると推論された箇所を重点的にデバッグとレビューを行うことで品質の向上に役立てている．一例として，メトリクスを求め不具合の起こる可能性が高いロジックの推論を行うことで，品質の向上に貢献してきた．最近では，メトリクスのような統計的手法だけでなく，機械学習及び深層学習を用いて，ソースコード片からソフトウェアの不具合の混入を推論する方法などがある．我々は，ソースコードを画像化して不具合の有無でカテゴリ分けを行い，深層学習を用いて学習させた．学習の結果，作成されたモデルを使用して，プログラムの不具合を推論した．本稿は，不具合の推論を行う上で課題であった推論精度の向上を試みた．これまで，全てのプログラムを学習させて一つのモデルを作成してきたが，本稿ではプログラマを条件によりグループ化し複数のモデルを作成した．条件は，プログラムを作成したプログラマの経験年数およびスキルを評価した結果を用いた．この条件を用いて評価の近いプログラマでグループ化している．グループ化したプログラムで，それぞれ学習を行い複数のモデルを作成した．同等のスキルを持ったプログラマが作成したプログラムは，コードの品質が揃うため，深層学習において不具合の特徴を捉えやすくなる．複数の学習モデルを用いて推論した結果を，集約して全体の推論結果とすることで，精度が向上すると考えたのである．推論の精度が向上することを確かめるため，実験を行った．

1. はじめに

これまで，システム開発におけるプログラムの品質を確保するため，開発者による自己レビューや設計者との対面レビューが行われてきた．年々システムの規模は増大するが，開発費用などのコストは抑えるといった状況にある．コストが抑えられると，ソフトウェア開発は品質に悪い影響が及びやすい．成果物であるソフトウェアの作成が優先され，品質管理が後回しにされた結果，品質向上を行うために必要な期間と費用を削減せざるを得ない状況に追い込まれることがあるからである．品質管理は，コスト削減によりこれまでと同じ手法では品質を保つことが難しくなってきている．

本稿では，畳み込みニューラルネットワーク（CNN）で作成したモデルを適用し，ソフトウェアの不具合の箇所を発見させる．このシステムを，CNN based bug inference system（略して CNN-BI system）と呼ぶ [12].

研究では，プログラムの不具合の傾向を予測するため，プログラマのスキルや経験年数が異なる複数のモデルによる推論の結果，精度が向上することを検証する．複数のモデルを使用して推論をすることは，深層学習の精度向上のために行われる．

研究の目的は、以下の問いに答えることである．

- 複数のモデルを深層学習により学習した結果，不具合の推論は精度が向上したか？
- 複数のモデルで推論ができた不具合とできなかった不具合はどのような内容であったか？
- 複数のモデルの推論結果と，実際に不具合対応した内容を比較した結果，課題は何であるか？

これらの研究課題に対する答えを得るためには，プログラムの作成者をスキルと経験年数により　グループ化し

学習する必要がある．プログラムのソースコードを色付きの要素を持つ画像に変換し，CNN-BI システムを用いて学習した．

我々は，これまでプログラムを単一のモデルで学習し，CNN-BI システムを用いて推論を行い，プログラムの不具合の傾向を推論できるかどうか，適合率と再現率と F 値で評価を行った [12]．本稿では，プログラマをグループ化して学習した複数のモデルを使用し，各モデル単位でプログラムの不具合の推論を行い，推論結果を多数決により集約し，プログラムの不具合を推論する．不具合の推論の結果，精度が向上することを検証した．

本稿は，以下の内容で構成されている．第 2 節は，関連研究を示す．第 3 節は，研究内容について説明する．第 4 節で，考察を示す．第 5 節で，まとめを行い，研究の結果と今後の課題を述べる．

2. 関連研究

これまで，ソフトウェアの不具合検出は，統計学的な手法により推論されてきた．Ruchika Malhotra [8] は，品質を推論するための評価基準は、ソフトウェア開発の初期段階でソフトウェアの品質を高めるために不可欠であると言っている．CK メトリクス [2] と QMOOD メトリクス [5] を使用して設計段階で障害の可能性を推定するモデルを作成する．機械学習の手法であるランダムフォレストとバギングを使用した．

また，近藤 [10] らは，コード分析による不具合推論では粒度が荒く不具合推論のフィードバックが遅いという問題を解決するために深層学習を適用している．Yang ら [7] は，変更に対するソフトウェアメトリクス [1] [3] [4] に深層学習を適用したが，ソースコード片に対して深層学習の方法の一つである畳み込みニューラルネットワーク (CNN) から W-CNN [7] を提案し推論が可能であることを示した．

森崎 [11] はソフトウェアの品質を管理するために，ソフトウェアの読みやすさの評価に深層学習を適用した．

本稿では，CNN-BI システムを用いて，バギングを考慮した複数の学習モデルを用いる推論が，プログラムの不具合予測で，精度が向上することを検証する．

3. 研究内容

我々は，プログラムのソースコードを画像に変換し，深層学習を用いてプログラムの不具合の有無でカテゴリ分けを行ったのち，教師あり学習をした．本稿では，教師あり学習を行うにあたり，プログラマのスキルを条件としてグループ化を行う．教師あり学習の対象となるプログラムは，グループに属するプログラマの作成した全てのプログラムを用いる．学習はグループ毎に行われるため，グループの数とモデルの数は同じである．学習結果のモデルを用いて，各グループのモデル単位にプログラムの不具合を推論する．学習と推論は，深層学習の手法の一つである畳み込みニューラルネットワークにより行われる．本稿の学習及び推論を行うために，Google の TensorFlow-gpu 1.12.0 に別途 KERAS 2.2.4 をインストールした環境を用いる．OS は，Windows10 HomeEdition 64bit である．ハードウェアの構成は，CPU : Corei7-6700K，Memory : 24GB，GPU : GeforceRTX2070Super で実験を行う．

3.1. 概要

本稿では，6 人のプログラマが作成したプログラム 75 本を使用する．プログラムは，VisualBasic2008 を用いて作成されている．このプログラム 75 本を，学習データ（学習に使用するデータと学習を検証するためのデータ）とするため，画像に変換する．プログラムを画像に変換する理由を以下に示す．

1. 画像から不具合を起こすプログラムの特徴を学習させる
   不具合を起こすソースコードは，コードの記述に特徴があるはずである．その形を学習させるために，ソースコードの画像を用いる．ソースコードの特徴を画像として識別できるようにするためソースコードの構成要素毎に色分けを行う．
2. プログラム作成中の不具合推論を可能にする
   ソースコードを画像化することで，コーディングの完了・未完了を問わず不具合の推論を行うことが出来る．

3.2. 研究のアプローチ

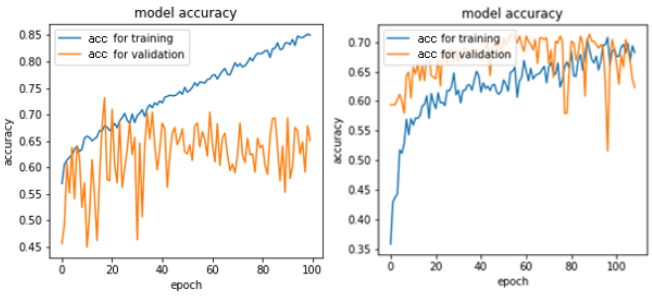
本稿の研究は，以下の手順で行う．

1. 学習データと推論データの収集.
   575 本のプログラムより，6 人のプログラマが作成した 75 本を選択した．この 75 本全てを学習に使用する．学習では，学習（training）と検証（validation）が行われ，学習データを 7 対 3 の割合で分割している．推論データは，学習に使用していないプログラムを，別途 5 本ランダムに選択した．この 5 本のプログラムの作成者は，学習データのプログラマとは異なるようにしている．すべての推論で，同じ 5 本のプログラムを用いる．

2. プログラムを画像に変換するための色分けを行う.

   - 命令文：青
   - 予約語：オレンジ
   - コメント：緑
   - モジュール名：明るい緑（太字）
   - グローバル変数：青（太字）
   - ローカル変数：黒
   - コントロール名：ピンク
   - ユーザ関数，サブルーチン名：赤
   - ユーザ定義名：ブラウン
   - 文字列：紫

3. 変換したプログラム画像より学習データを作成.
   プログラムのソースコードを 30 行単位に分割して画像に変換する．フォントは MS ゴシックで 6.8pt であり色分けがされた箇所は Bold とした．1 行は 135 文字であり，1 行に収まらない場合は行を折り返す.
   ソースコードの画像化にあたり，画像化のためのツールを作成した．不具合の修正前と修正後のソースコードを用意して，プログラムの内容を比較する．プログラムの相違のある箇所が，どのカテゴリに属するか判断し，ソースコードを画像化する．画像化されたソースコードは，画像 1 枚単位にカテゴリ分けされたフォルダに格納される．ツールを使用することで，プログラム 75 本全ての学習データの作成が 1 時間程度で出来た．手作業で作成した場合，27 本で 50 時間程必要としていたため，学習データ作成の短縮が可能となった.



Flattenを使用　　Global Average poolingを使用

図 1. 学習経過の違い

4. CNN-BI システムを使用して学習を行う.
   学習には，学習済みデータセットである VGG16 [6] を使用した．VGG16 の，最後の畳み込み層である BLOCK5 と全結合層を，研究で使用する学習データで学習する．全結合層には，Global Average Pooling(GAP) [9] を使用した．Flatten [6] を使用する方法と比較した学習の経過の違いを，図 1 に示す．図 1 のグラフは，縦軸が適合率，横軸が学習回数である．青い線は学習データのうち学習に使用したデータの適合率，オレンジの線は検証に使用したデータの適合率である．青い線とオレンジの線が共に右上がりのグラフが良い学習結果であり，過学習ではオレンジの線である検証用データの適合率が向上しない.
   Global Average Pooling を使用した学習モデルが，Flatten を使用する方法より過学習を抑制していると判断した．プログラムを画像化して学習する場合に，学習データのうち検証に使用したデータ（validation）の適合率が向上していることが図 1 より確認できるためである.
   学習のパラメータは以下の通りに設定した.

   - 最終結合層のモデル
     `GlobalAveragePooling2D()(x)`
     `Dense(256, activation='relu')(x)`
     `Dropout(0.5)(x)`
     出力は、softmax 関数を使用する.
   - モデルのコンパイルパラメータ
     `batch_size : 16`
     `optimizer:SGD(lr=0.00002, momentum=0.9, decay=1e-6, nesterov=True`

loss：categorical_crossentropy

5. 学習と推論.
本稿の学習と推論は，以下の手順で行う.

(1) プログラマの経験年数とスキルを条件として，3 つのグループに分ける.

(2) 3 つのグループが，それぞれ学習を行う.

(3) 学習した結果、作成された 3 つの学習モデルを，学習に使用していないプログラムを用いて不具合の有無を推論させる.

(4) 不具合を推論した結果を，多数決戦略により結論を得て，評価を行う.

(5) 評価結果から多数決戦略の修正を行い，再度評価する．評価は，適合率と再現率と F 値で行う．F 値は推論精度の指標である．適合率と再現率はトレードオフの関係にあるので，F 値を用いて精度を評価する.

3.3. 学習モデルの作成方法の見直し

学習モデルを作成するため，6 人が作成したプログラム 75 本を使用する．各プログラマが作成したプログラムの本数を，表 1 に示す.

表 1. 各プログラマの作成した本数

作成者	作成本数
pgmA	37
pgmB	11
pgmC	6
pgmD	7
pgmE	8
pgmF	6

表 1 のプログラムの本数は，プログラマにより本数に違いがある．続いて，6 人のプログラマの経験年数とスキルを表 2 に示す.

経験年数とスキルは，システム開発当時のプロジェクト内部の評価である．プログラマを経験年数とスキルで比較した結果，プログラマ 6 人のスキルに違いがあることが分かる．経験年数は，必ずしもスキルと一致していない．プログラマをグループ化するため，プログラマの

表 2. プログラマ毎の経験年数とスキル

作成者	経験年数	スキル (品質)
pgmA	15 年	C
pgmB	5 年	B
pgmC	6 年	B
pgmD	10 年	A
pgmE	4 年	A
pgmF	12 年	A

スキルを条件としてグループを分けた．スキルの近いプログラマであれば，グループに所属するプログラマが作成するプログラムは同程度の品質となると考えられる．グループ化した結果，学習モデルは 3 つのグループに分割した．プログラマ pgmA をグループ 1(Mdl1)，プログラマ pgmB と pgmC と pgmE をグループ 2(Mdl2)，プログラマ pgmD と pgmF をグループ 3(Mdl3) としてグループ化した．学習データは，各グループに所属するプログラマの作成したすべてのプログラムを使用する.

3.3.1 学習カテゴリの見直し

本稿では，学習カテゴリの分類方法を見直す．不具合の有無のみでカテゴリを分けるのではなく，不具合の分類でカテゴリを増やす．不具合の分類単位で，ソースコードの見え方が変わることにより，ソースコードの画像から特徴を捉えやすくなるのではないかと考えた．カテゴリ分けの結果を表 3 に示す.

表 3. 学習のカテゴリ

カテゴリ	カテゴリ名	不具合の内容
OK	不具合なし	
NG1	制御の修正	IF 文など制御構文の修正
NG2	その他修正	代入、SQL 文などの修正
NG3	ロジック追加	ロジックが挿入された
NG4	ロジック削除	ロジックが削除された

表 3 のカテゴリ分けを使用して学習を行う．学習を行ったグループ 1 の学習経過を図 2 に示す.

図 2 のグラフは，右図は図 1 と同じく適合率（accuracy）のグラフである．左図は損失 (loss）のグラフとな

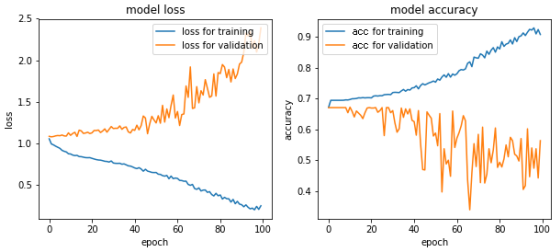


図 2. NG を 4 カテゴリにした場合の学習経過

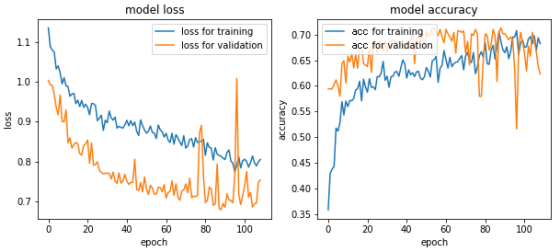


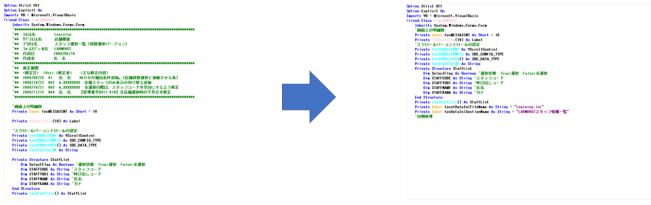
図 3. NG を 2 カテゴリにした場合の学習経過

る．縦軸は損失（loss）であり横軸は学習回数（epoch）である．
学習の経過は，学習データである学習用データ（training）および検証用データ（validation）が共に，損失（loss）は右下がりで適合率（accuracy）は右上がりになることで，過学習を起こさず正常に学習がされていると判断することができる．学習の経過を確認したところ，検証用データの損失と適合率が過学習の傾向を示した．正常に学習させるため，カテゴリ分けの再検討を行う．4 つの不具合の種類により分けたカテゴリのうち，2 つは不具合を記述の種類で分けている．残りの 2 つは，ロジックの挿入と削除であるため，記述の内容に関わらずカテゴリ分けが行われている．カテゴリ分けは，ソースコードの記述の違いで行った方が良いと考え，2 つのカテゴリに分けた．ソースコードの記述を画像化して学習データとするからである．カテゴリ分けを変更した結果を表 4 に示す．

表 4. 修正後の学習のカテゴリ

カテゴリ	カテゴリ名	不具合の内容
OK	不具合なし	
NG1	制御の修正	IF 文など制御構文の修正
NG2	その他修正	代入、SQL 文などの修正

不具合を，記述の種類で分けるという同じ基準でカテゴリを分けることで，学習データの画像の形も同じ傾向になると考えた．プログラムの記述の違いでカテゴリ分けをすると，画像化したソースコードの画像の違いがそのままカテゴリ分けとなる．2 つのカテゴリの内訳は，表 4 にある制御の修正とその他の修正である．カテゴリを変更した後，学習を行ったグループ 1 の学習経過を図 3 に示す．



リファクタリング前　　リファクタリング後

図 4. リファクタリングのイメージ

カテゴリを変更したことで，過学習の抑制が出来ている．学習データの特徴をより捉えやすくなった結果と判断した．

3.3.2　プログラムを画像化する方法の見直し

プログラムを画像化する際に，画像の内容を均一化するため，プログラムのソースコードをリファクタリングした．画像の内容を均一化することで，画像の情報量を増やし，より特徴を捉えやすくできないか考えたからである．リファクタリングのルールを以下に示す．

- ヘッダコメントの除去
- 空行の除去
- インデントは変更しない

リファクタリング前後の画像を図 4 に示す．図 4 では，左図がリファクタリング前であり，右図はリファクタリング後である．リファクタリングを行うことで，ヘッダコメントと空行が詰められている．

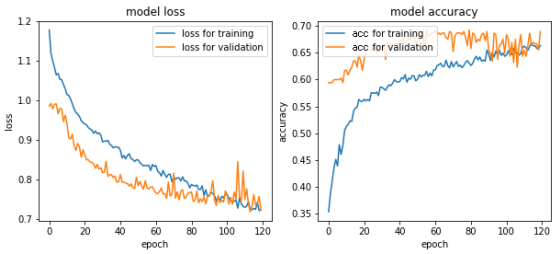


図 5. 学習経過

3.3.3 学習結果

プログラマのグループ化，学習カテゴリの見直し，プログラムを画像化する方法の見直しを行ったのち，学習データによる教師あり学習を行う．学習の経過を，グループ 2 を例に図 5 に示す．学習の経過から，学習データである学習用データ（training）および検証用データ（validation）が共に，損失（loss）は右下がりで適合率（accuracy）は右上がりになることで，過学習を起こしていないことを確認した．確認したのち，すべてのグループの学習を行う．3 つのグループが学習した件数と学習結果を，表 5 に示す．表 5 の OK 枚数と NG1 枚数と

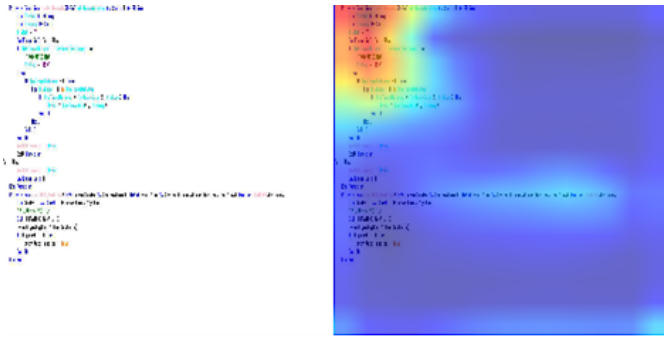
表 5. 学習件数と学習結果

グループ	OK 枚数	NG1 枚数	NG2 枚数	学習結果の適合率
Mdl1	731	405	502	71 %
Mdl2	386	103	83	70 %
Mdl3	219	48	34	81 %

NG2 枚数は各学習カテゴリの枚数である．OK は不具合なし，NG1 はロジックの不具合，NG2 はスクリプトの不具合である．

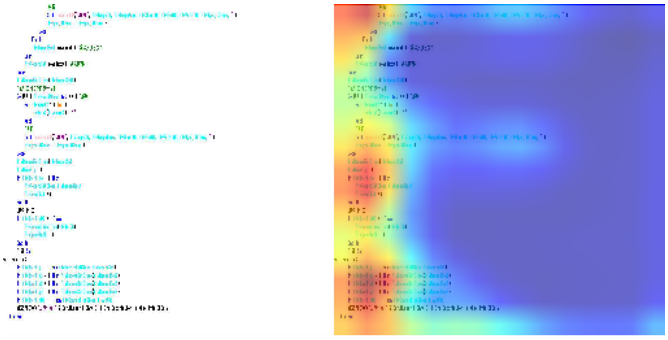
3.3.4 学習結果の視覚化

深層学習の結果をヒートマップ [13] [14] により視覚化した．視覚化により，学習モデルが画像のどの部分に着目しているかを確認することができる．記述の誤りを教師あり学習のカテゴリとしたことで，着目している箇所を目視して，カテゴリ分けの通りに学習していると判断できると考えた．以下に，可視化した結果を示す．



ソースコード　　ソースコード＋ヒートマップ

図 6. カテゴリ：OK の場合

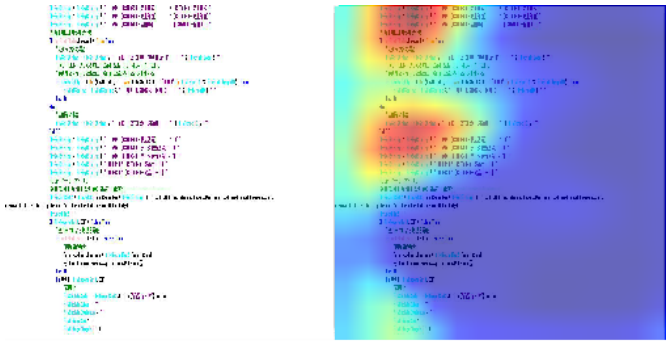


ソースコード　　ソースコード＋ヒートマップ

図 7. カテゴリ：NG1 の場合

- カテゴリ：OK
  「不具合のない場合」のヒートマップ例を図 6 に示す．
- カテゴリ：NG1
  「IF 文など制御構文の修正」のヒートマップ例を図 7 に示す．
- カテゴリ：NG2
  「代入、SQL 文などの修正」のヒートマップ例を図 8 に示す．

左図がソースコード，右図がヒートマップである．ヒートマップは着目しているほど赤色に近くなる．ヒートマップで確認した結果，カテゴリが NG1 であればロジックを全体的に，カテゴリが NG2 であればスクリプトである文字列（紫色）を中心に着目している．カテゴリ NG1 はロジックの誤りを示すカテゴリで，カテゴリ NG2 はスクリプトの誤りを示すカテゴリである．各カテゴリ単



ソースコード　　ソースコード＋ヒートマップ

図 8. カテゴリ：NG2 の場合

位に，ヒートマップをランダムに 10 例取得して確認を行った結果，全件ではないが取得した範囲で同じ傾向であったため，学習の経過に誤りは無いと考えた．

3.4. 学習モデルによる推論

深層学習により作成した，3 グループの学習モデルを使用して推論を行う．

3.4.1　推論結果

3 つのグループがそれぞれ推論した結果を集計する．

- グループ 1 の推論結果を，表 6 に示す．

表 6. グループ 1 の推論結果

ID	NG1	正解	NG2	正解	適合率	再現率	F 値
A	0	0	8	8	100 %	67 %	0.8
B	0	0	2	2	100 %	100 %	1
C	0	0	21	9	43 %	47 %	0.45
D	0	0	5	0	0 %	0 %	0
E	2	0	53	37	67 %	49 %	0.57

ID　　：プログラムの ID
NG1　：カテゴリ NG1 と判断した枚数
正解　：NG1 のうち実際に起きた不具合と一致した枚数
NG2　：カテゴリ NG2 と判断した枚数
正解　：NG2 のうち実際に起きた不具合と一致した枚数

- グループ 2 の推論結果を，表 7 に示す．

表 7. グループ 2 の推論結果

ID	NG1	正解	NG2	正解	適合率	再現率	F 値
A	0	0	6	6	100 %	50 %	0.67
B	1	0	6	2	29 %	100 %	0.44
C	7	2	16	6	35 %	42 %	0.38
D	11	0	12	0	0 %	0 %	0
E	9	3	49	27	52 %	40 %	0.45

- グループ 3 の推論結果を，表 8 に示す．

表 8. グループ 3 の推論結果

ID	NG1	正解	NG2	正解	適合率	再現率	F 値
A	7	1	3	3	40 %	33 %	0.36
B	4	0	1	1	20 %	50 %	0.29
C	5	0	8	3	23 %	16 %	0.19
D	36	2	5	0	5 %	1 %	0.01
E	60	14	36	29	45 %	57 %	0.5

- 各グループが決定したカテゴリを多数決した推論結果を，表 9 に示す．

表 9. 各グループが決定したカテゴリを多数決した推論結果

ID	NG1	正解	NG2	正解	適合率	再現率	F 値
A	0	0	6	6	100 %	50 %	0.67
B	0	0	2	2	100 %	100 %	1
C	0	0	13	5	38 %	26 %	0.31
D	1	0	4	0	0 %	0 %	0
E	2	1	38	34	88 %	47 %	0.61

- 各グループの推論した割合の数値を集計し，多数決した推論結果を，表 10 に示す．

3.4.2　推論結果の比較

推論結果は以下の 3 通りを取得する．

- グループ単位の推論結果
  推論した結果は，各カテゴリ毎に 0 から 1 の範囲で

表 10. 全体を多数決した推論結果

ID	NG1	正解	NG2	正解	適合率	再現率	F 値
A	0	0	7	7	100 %	58 %	0.74
B	0	0	2	2	100 %	100 %	1
C	0	0	16	9	56 %	47 %	0.51
D	1	0	2	0	0 %	0 %	0
E	1	0	44	38	84 %	51 %	0.63

出力される．一番大きい数値を出力したカテゴリを推論結果とした．

- 各グループの推論した結果を多数決する
  グループ 1 が OK，グループ 2 が NG1，グループ 3 が NG1 と推論した場合，多数決で NG1 が推論結果となる．全てのグループが，別のカテゴリを推論した場合は OK として扱う．
- 各グループの推論したカテゴリの割合を多数決する
  各グループが推論した結果より，出力した数値をカテゴリ単位に全て加算する．一番大きい数値となったカテゴリを推論結果とする．

グループ単位の推論結果では，グループが異なると学習に使用したプログラムも異なるため，推論結果に違いが出る．全グループを多数決した結果，適合率はそれぞれのグループの良い結果を集約したように向上している．多数決は，各グループの推論結果を多数決するより，各グループの推論した割合の数値を集計する方法が，再現率および F 値で良い結果となる．

不具合の推論で重要となる再現率は，多数決をする場合とそうでない場合で，大きな変化はない．ただ，必ずしも各グループは同じ推論結果を出していない．推論結果に誤りの多いカテゴリ NG1 に，不具合は無い（カテゴリ OK）と判断していることが多かった．

しかし，各グループのカテゴリ NG1 の推論結果に，NG1 の推論結果の数値がほかのカテゴリ OK となった推論結果より，高い値を示す少数意見があることに着目した．このカテゴリ NG1 の推論結果を集計した値に，閾値を設定する．閾値は，推論が誤っている場合に，正解となるカテゴリ NG1 の平均値である 0.7 とする．推論に使用したプログラムに限定されるという前提であるが，0.5 から 1.5 まで，平均値を中心に 0.1 刻みで閾値を変更したが平均値である 0.7 のときの再現率と F 値が最も良い結果を示している．閾値を設定することで，精度に変化があるか確認するため，閾値を超えた推論結果がある場合にカテゴリ NG1 の可能性があるとして推論した結果を，表 11 に示す．

表 11. 多数決で少数意見を含めた推論結果

ID	NG1	正解	NG2	正解	適合率	再現率	F 値
A	8	3	7	7	67 %	83 %	0.74
B	10	0	2	2	17 %	100 %	0.29
C	28	5	16	9	32 %	74 %	0.44
D	92	10	2	0	11 %	71 %	0.19
E	139	30	44	38	37 %	91 %	0.53

表 11 の再現率は，表 10 より向上する．しかし，F 値が大幅に下回る結果となった．

他の手法による精度向上の実現のため，各グループの推論結果を多数決するときの重みとして，プログラマのスキルを推論結果に反映させた．プログラマのスキルは，表 2 より 3 段階で評価している．このスキルを，段階ごとに A が 3 点，B が 2 点，C が 1 点で評価し，各グループの平均値を集計した．集計した結果を，表 12 に示す．

表 12. 各グループのスキル値と決定した重み

グループ	平均スキル (品質)	平均点数	重み
Mdl1	C	1.0	0.7
Mdl2	B	2.3	1.0
Mdl3	A	3.0	1.3

プログラマのスキルを考慮して，決定した重みを乗算し多数決した推論結果を，表 13 に示す．

表 13. 多数決でプログラマのスキルを考慮した推論結果

ID	NG1	正解	NG2	正解	適合率	再現率	F 値
A	0	0	7	7	100 %	58 %	0.74
B	0	0	2	2	100 %	100 %	1.0
C	0	0	13	9	69 %	47 %	0.56
D	4	0	4	0	0 %	0 %	0.0
E	3	1	43	37	83 %	51 %	0.63

表 11 と同じく，閾値を超えた推論結果がある場合に，

カテゴリ NG1 の可能性があるとして，推論結果に含めた場合の推論結果を，表 14 に示す．

表 14. スキルと少数意見を考慮した推論結果

ID	NG1	正解	NG2	正解	適合率	再現率	F 値
A	8	3	7	7	67 %	83 %	0.74
B	10	0	2	2	17 %	100 %	0.29
C	28	5	13	9	34 %	74 %	0.47
D	88	9	4	0	10 %	64 %	0.17
E	136	32	43	37	39 %	92 %	0.54

4. 考察

ソースコードを画像化して，プログラムの不具合を推論した．プログラムの不具合とソースコードの記述の見た目に関連があり，画像化し教師あり学習を行うことで，プログラムの不具合を推論できると考えたためである．不具合の推論結果は，推論を行ったプログラムで精度にばらつきがあった．本稿では，精度のばらつきが，すべてのプログラマの作成したプログラムを一つの学習モデルとして学習させたため，不具合の特徴を捉えにくかったのではないかと考えた．一つの学習モデルでは限界のあった推論の精度を向上させるため，6 人のプログラマが作成したソースコードを，プログラマのスキルにより 3 つのグループに分けて教師あり学習を行った．学習の結果，各グループが学習したモデルを使用し，不具合の推論を行った．推論した結果を用いて，多数決により不具合の有無を決定することを試みた．不具合の有無を，実際に発生した不具合の結果と比較し，推論が正しいことを検証した．

4.1. 複数のモデルによる推論

プログラマのスキルでグループ化した複数のモデルによる推論は，教師あり学習に使用するソースコードが異なるため，それぞれ異なる推論結果を示した．しかし，同一プロジェクトのプログラムを使用した推論であるため，精度の高い推論結果と精度の低い推論結果は，推論したプログラムの単位に同じ傾向を示している．各グループの推論結果を多数決すると，各グループの推論結果を適合率が高い方に集約することができた．しかし，プログラマを 3 つのグループに分けたことで，各グループの学習データの件数が減少した．少ない学習データを用いた検証方法は課題となる．

4.2. 推論の精度は向上したか

複数のモデルによる推論結果を，多数決で全体の推論結果とする手法は，適合率の向上が確認できた．しかし，不具合の推論では再現率の向上が必要である．本稿では，多数決で推論結果を決定しているため，少数意見となる推論結果があった．少数意見を汲み上げるよう推論結果を修正すると，再現率が向上した．ただし，そのまま推論結果としてしまった場合，対象となる不具合の数も増加するため，適合率と F 値が減少する．少数意見として，注意喚起をするなどの手法により，レビューを行う際の判断の助けにするような方法が良いのではないかと考えた．また，ロジックの修正であるカテゴリ NG1 の推論が，全体的に精度が低い結果となっている．ロジックの修正による不具合を推論することは，複数のモデルで推論し，多数決で推論結果を修正したとしても精度を向上させることができなかった．学習結果のヒートマップではロジックに着目しているが，ロジック修正のカテゴリである NG1 の推論は，不具合なしのカテゴリ OK と判断されることが多かった．ロジックの修正による不具合を，より高い精度で推論することは，今後の課題となる．

5. まとめ

本稿では，6 人のプログラム作成者を，経験年数とスキルが近い 3 つのグループに分けた．全てのグループで，個別に教師あり学習を行い，作成した 3 つの学習モデルを用いて，各モデルに同じプログラムを推論させた．全ての推論結果を使用し，多数決によって集約した推論結果を求め，精度が向上することを検証した．検証結果から，単一のモデルで推論した結果と比較して，すべてのモデルで多数決をとることで，適合率が向上した．さらに，推論結果の少数意見を取り入れることで，完全ではないがプログラムの不具合を推論する再現率を向上させることができた．本稿の問いに対する答えを以下に示す．

- 複数のモデルを深層学習により学習した結果，不具合の推論は多数決をすることで，適合率を向上させた．

- SQL 文などの埋め込みスクリプトの推論ができているが，ロジック修正の推論の精度が低かった．
- 複数モデルの推論の精度は，各モデルの推論結果の多数決をとり，加えて推論結果の少数意見を汲み上げることで，限定的に再現率を向上させることができた．再現率向上のため，ロジックの修正による不具合の推論は課題である．

我々が作成した推論のための学習モデルは，ロジック修正のような正常なプログラムと判別が難しい不具合の推論が課題である。課題の解決のため，プログラムの不具合の場所を特定する必要がある．CNN-BI システムは，1 枚の画像に収められたソースコードの範囲で，プログラムの不具合を推論するため具体的な箇所の特定が難しい．ロジック単位で推論することにより，推論が難しい不具合についてもロジック単位の学習で精度を改善できる可能性がある．今後，不具合のあるロジックを特定させるため，プログラムのロジック単位に，図やモデルなど画像の形を変形し，学習データを作成することで推論の精度を向上させたい．

参考文献

[1] W.Li, W.Henry: Object-Poiented Metrics that Predict Maintainability, In Journal of Sytems and Software, Vol.23, Issue.2, pp. 111-122, Nov. 1993.

[2] S.Chidamber, C.Kemerer: A Metrics Suite for Object-Oriented Design, IEEE transactions on Software Engineerring, Vol.20, Issue.6, pp. 476-493, June 1994.

[3] M.Lorenz, J.Kidd: Object-Oriented Software Metrics, Prentice-Hall Inc., July 1994.

[4] F.Brito e Abreu, W.Melo: Evaluating the Impact of Object-Oriented Design on Software Quality, 3rd IEEE International Software Metrics Symposium, pp. 90-99, Mar. 1996.

[5] J.Bansiya, C.G.Davis: A Hierarchical Model For Object Oriented Design Quality Assessment, IEEE transactions on Software Engineerring, Vol.28, Issue.1, pp. 4-17, Jan. 2002.

[6] Karen Simonyan, Andrew Zisserman: Very Deep Convolutional Networks for Large-Scale Image Recognition, CoRR, pp. 1409-1556, Apr. 2015.

[7] X.Yang, D.Lo, Y.Zhang, J.SUN: Deep Learning for JustInTime Defect Prediction, 2015 IEEE International Conference on Software Quality, pp. 17-26, Aug. 2015.

[8] R.Malhotra, A.Jain: Fault Prediction Using Statistical and Machine Learning Methods for Improving Software Quality, Journal of Information Processing Systems, Vol.8, No.2, pp. 241-262, June 2012.

[9] Min Lin, Qiang Chen, Shuicheng Yan: Network In Network, International Conference on Learning Representations 2014, Mar. 2014.

[10] 近藤将成, 森啓太, 水野修, 崔銀恵: 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，pp. 35-45, Aug. 2017.

[11] 森崎 雅稔: ディープラーニング活用事例と使いこなしの勘所 :［最適化・推論分野］6．AI によるソースコードのレビュー -ディープラーニングでコードの美しさを診断する-, 情報処理, Vol.59, number.11, pp. 985-988, Oct. 2018.

[12] K.Ogawa and T.Nakatani: Predicting Fault Proneness of Programs with CNN, 11th International Conference on Agents and Artificial Intelligence, Vol.1, pp. 321-328, Feb. 2019.

[13] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, Dhruv Batra: Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization, International Journal of Computer Vision, Vol.128, Issue4. pp. 336–359, Apr. 2020.

[14] François Chollet: Python と Keras によるディープラーニング , 株式会社マイナビ出版, 第 5 章, pp. 166-186, Dec. 2018.