CNN-BI システムによるプログラムの不具合発見の検証

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．従来の方法には，ソフトウェアメトリクスを用いてソフトウェアシステムの品質を評価する方法がある．本稿では，プログラムの不具合を有無を推論するために、深層学習技術の *1* つである *CNN*（*Convolutional Neural Network*）を適用する．不具合を起こすプログラムの記述には共通点があるのではないかと考えた．プログラムの記述を目視すると，関数やサブルーチンの集合体でインデントにより模様に見える．記述に共通点があるのであれば，見た目の模様にも共通点があると考えられる．プログラムの構文の文法に従い，文字に色付けすることでより特徴付けられる．プログラムの記述の特徴を，深層学習を用いて，ソースコードの構文の形から不具合を推論することを試みた．これには，ソースコードを画像化し，深層学習により不具合の推論を行った．推論結果を検証するために，不具合のある画像を推論した結果と，推論用プログラムの中で実際に発見された不具合の内容を比較した．この比較実験では，画像化したプログラムを使用して学習用データ及び推論用データを作成した．作成したデータを用いて，学習と検証および推論することで実験を行った．

1. はじめに

近年，高機能なシステムが求められるようになってきた．そのためシステムの規模は大きくなる方向にある．その半面，システム開発は低コストを求められるようになった．それにより，開発費用といったコストを抑える努力が，プロジェクト管理者に求められるようになってきた．プロジェクト開発者は，コストを抑えるために，早期にプログラムの不具合を発見し解決する必要がある．システム開発の手法，開発支援ツールは飛躍的に進歩しているが，システム開発を行うのは人間である．しかし，人間は必ずミスをする．ミスを見逃すとシステムは障害を起こし，システムの供給者及び利用者に多大な損害を発生させる．これまで以上にシステムの品質を向上させることの重要性が増している．研究者や開発者は，分析や設計，実装における人的ミスに対処するための技術を開発した．開発の初期段階で，これらの不具合を見つけて修正することが重要な目標の 1 つである．
ソフトウェアの複雑さを軽減するためのシステム構築方法が提案されている．オブジェクト指向ソフトウェア構築方法論 [1] が効果的であるとしても，信頼できるソフトウェア構築方法は，ソフトウェア内の不具合を見つけることにある．結局のところ，ソフトウェアの構築は人間の能力に依存している．実際には，テストですべてのエラーや欠陥を修正できるわけではない．さらに，未知の不具合によって，致命的な障害が発生することもある．本稿では，ソフトウェアの品質を向上させるために CNN（Convolutional Neural Network）モデルを適用する．このシステムを，CNN based bug inference system（略して CNN-BI system）と呼ぶ．

本稿の目的は，プログラムの不具合の傾向を予測するため，CNN アプローチの有効性を提示することである．CNN は深層学習の 1 つであり，画像認識と画像に基づく推論の分野で成功している．我々は，CNN-BI システムが，深層学習を用いてプログラムの不具合の特徴を抽出することを期待している．CNN によるアプローチの有効性を以下に示す．

- 不具合を引き起こすソースコードは，コードの記述に特徴を持っている．
  CNN のアプローチは，白黒の画像ではなく，文字

に色付けを行ったプログラム画像を用意することで，より特徴を捉えやすくなる．

- CNN モデルがプログラムの不具合の可能性を推測できることで，コーディングの完了・未完了を問わずプログラムの不具合を確認して修正することができる．

プログラムの不具合の傾向を推論するため，ソースコードの画像を利用し，CNN-BI システムを用いて CNN モデルで学習する．

研究の目的は、以下の問いに答えることである．

- 画像を用いた深層学習の結果，不具合の推論は可能であったか？
- 推論ができなかった不具合はどのような内容であったか？
- 不具合の精度は実際に不具合対応した結果を比較してどうであり，また精度を上げるにはどうすれば良いかを今後の課題と併せて提示する．

これらの研究課題に対する答えを得るためには，プログラムの要素を分類し，プログラムを CNN-BI システムが読む画像に変換する必要がある．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．学習は，学習データの件数，学習時間などの学習コストを軽減するために，転移学習 [2] を適用した．推論の目的は，プログラムが不具合を起こしやすいかどうかを推論することであり，教師あり学習を CNN モデルの学習方法として選択した．

学習の後，CNN-BI システムを用いて，プログラムの不具合の傾向を推論できるかどうか評価を行った [3]．本稿では，さらに，適合率について精度を向上させるための方策を検討し，検証を行った結果を報告する．

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 節では，研究内容について説明する．第 4 節では，考察を示す．第 5 節は，まとめを述べる．

2. 関連研究

ソフトウェアの品質を向上させる方法には，様々な方法がある．統計的手法はソフトウェアの品質を視覚化することができる．統計的方法を利用する技術が広く適用されている．定性的情報を定量的に提示することができれば，我々はプログラムの定性的特徴を定量的データと比較することができる。ソフトウェアの品質を向上させるために，ソフトウェアの品質を定量的に観測および管理することができる．

Cyclomatic complex は，1976 年に McCabe [4] によって提案された．循環的複雑度と欠陥数の間には強い相関係数がある．したがって，プログラムの循環的複雑度が大きいと測定される場合，プログラムは，循環的複雑度が小さいプログラムよりも複雑であることを意味する．定量的に提示された「比較的複雑な」プログラムは慎重に検討する必要がある．10 年後，複数のパラダイムが適用され始めた．オブジェクト指向ソフトウェアのために，Chidamber と Kemerer [5] は，結合と結合，読みやすさ，理解しやすさ，修正可能性などに基づいて複雑さを測定するための測定基準を導入した．それらのメトリクスのセットは，*CK metrics* として知られている．Zimmermann [6] はまた，欠陥予測モデルを導入した．これらの方法は、ソフトウェアシステムの品質を評価するための定量的アプローチに基づいてる．

定量的アプローチには評価基準が必要である．Lorenz と Kidd [7] は，オブジェクト指向プログラムの品質の基準として典型的な測定基準についての経験的なデータを発表した．彼らの意図は，プログラムの測定値が「通常の」範囲内にない場合，プログラムを見直さなければならないということであった．これは，メトリクスがプログラムの不具合を見つけられないことを意味する．さらに，テストはプログラムの品質を向上させるのに効果的であるが，すべての不具合を発見することはできない．プログラムの品質を向上させる上で困難な点の 1 つは，複雑なプログラムではすべての不具合が発生するわけではないということである．どの部分を集中的に検証するかを決めるために，別のアプローチを採用した．CNN アプローチは，プログラムの複雑さだけでなくプログラムの構造的特徴によっても起こる不具合を見つけることができるかもしれない。

google cat [8] 以降，深層学習の研究と応用例が増えている．ソフトウェアの品質評価に，深層学習を適用した研究がある．たとえば，森崎 [9] 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．近藤ら [10] は，静的コード分析からのフィードバックの遅れに対処するため，深層学習を利用した．Yang ら [11] はまた，ソフトウェアの修正箇所を検証するため，深層学習を適用し，CNN ではなく

W-CNN を提案した．CNN-BI システムが，プログラムの不具合の可能性を予測できることを検証する．

3. 研究内容

深層学習を用いた，プログラムの不具合の推論を評価するために，推論の有効性を定性的および定量的に分析する．本節では，プログラムの不具合を推論するために，CNN が適用された事例について考察する．CNN-BI システムの学習は，OS は Windows10 HomeEdition を使用し，Google から提供されている TensorFlow と KERAS を利用し，Python で学習プログラムの開発を行う．開発環境の構成は，CPU : Core i7-6700K，MainMemory : 8GB，GPU : GeforceGTX980ti をそれぞれ使用する．

3.1. 概要

本稿で学習及び推論に使用したプログラムは，15 名の開発者によるプロジェクトであり，プロジェクトの開発期間は 1 年間であった．プロジェクトで開発されたプログラムは VisualBasic2008 により開発され，プログラム本数は 575 本であった。この 575 本より，CNN-BI システムを訓練するための学習用データ及び学習経過を検証するための検証用データ，訓練後に検証を行うための推論用データを取得する。

学習までの手順を以下に示す．

1. 学習データの収集.
   575 本のプログラムより，27 本をランダムに選択した．

2. プログラムのステートメントの各要素は，次のカテゴリに基づいて色分けを行う．
   - 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. 各プログラムを最大 30 行のコードでサブプログラムに分割した．135 文字以上で行を折り返す．フォントは MS ゴシックで 6.8pt，文字色が設定された場合は Bold とした．

4. プログラムの画像は，B5 サイズ横で作成する．
   学習用データとして 1490 枚の画像を取得した．言い換えれば，各画像はプログラムの断片を表している．不具合の傾向は，各画像について学習され推論される．

5. 教師あり学習の学習データは 2 つのグループに分類される．“不具合あり” と “不具合なし” は，それぞれ “BUG” と “OK” でラベル付けされる．

6. CNN-BI システムより学習用データを用いて学習した.
   CNN-BI システムの学習モデルとして，ImageNet の学習モデルである VGG 16（VGG ILSVRC 16 層）を使用した．VGG16 には，13 の畳み込み層と，3 つの全結合層を含む合計 16 の層がある．VGG16 は 1000 のカテゴリの学習済みデータセットである．
   図 1 に，VGG16 の構成を示す．畳み込み層は，CNN-BI システムの学習モデルとして再利用した．プログラムの不具合を推論するために，全連結層を学習用データで訓練した．次に訓練したモデルを使用して，不具合の有無に基づき 2 つのカテゴリーに分類を行い，推論結果を出力した。図 2 に，我々が訓練した全結合層のモデルを示す．このような学習方法を転送学習と呼ぶ．転移学習は，特定の領域から，限られた量のデータしかない別の領域に学習済みモデルを適用することに役立つ．我々は転移学習を利用し，限られた量の学習データでソースコードの不具合を推論した．

7. 同様に，検証データとして 322 枚の画像を作成した．

8. 検証データを用いて CNN-BI システムを検証した．
   学習過程の傾向を検証し，未学習および過度な過学習でないことを確認した．

学習および検証データの数を表 1 に示す．

表 1. 学習および検証データ数

Data Category	Label	#cases
Training data	OK	945
	BUG	545
Verification data	OK	213
	BUG	109

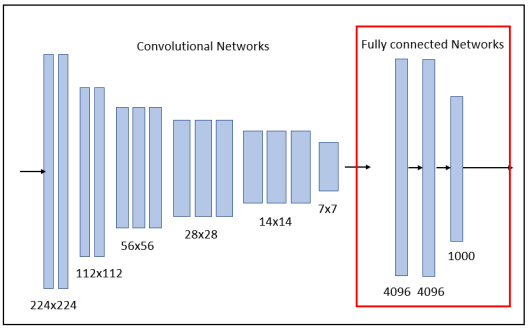


図 1. VGG16 モデルの構成.

3.2. 学習結果

学習データを用いて学習した結果，85%の精度を達成することを目標とする．学習を行う際に，2 つのカテゴリのデータ数が偏りなく学習できるよう．学習データに重み付けを行った． [12] 学習データを用いて実施した学習の過程を図 3 に示す.

左図の x 軸と y 軸は，それぞれエポックと損失を表す．同様に，右図の y 軸は学習の精度を表す．ここで，「エポック」とは、学習のサイクルである．例えば，100 エポックは，学習が 100 回繰り返されたことを意味する．

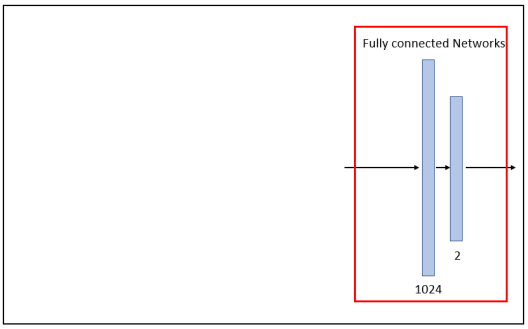


図 2. 全結合層の修正結果

図の青線は学習時の損失と精度を表す．赤線は検証時の損失と精度を表す．我々は，100 エポックの学習を行った．図 3a に示すように，損失は右下に減少している．図 3b は精度を表し，右上に上昇している．学習の結果，損失と精度はそれぞれ 19.04% と 82.97% に達した．

学習結果の精度の目標は当初，85% 以上であった．実際の学習結果の精度は 82.97% で，目標としていた精度より低い．しかし，学習経過より過度な過学習及び未学習ではないと判断し，学習結果のモデルを使用することとした．

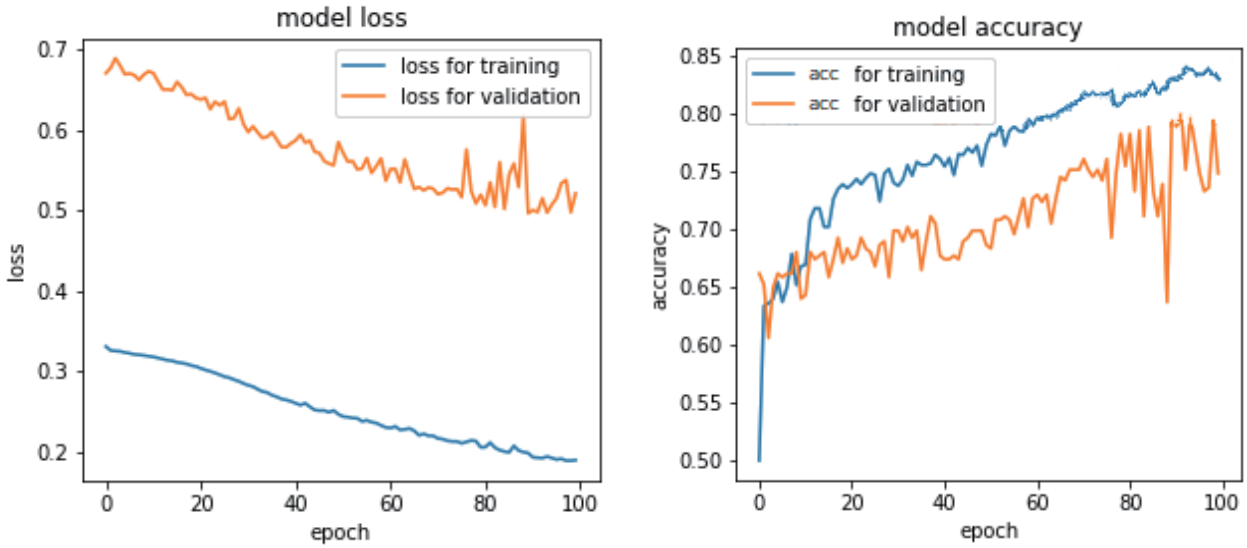
3.3 推論結果

我々は訓練された CNN-BI システムを使用し，プログラムの不具合を推論した．推論の条件は以下の通りであった．

1. CNN-BI システムの学習に使用したソースコードは，不具合の推論には使用しない．学習データと推論データは異なるプログラムを使用し，同じ画像を学習と推論で使わない．
2. 不具合の推論に使用するソースコードは，不具合を解決する前のプログラムを使用する．
3. ソースコード「OK」と「BUG」の画像の数は，学習データの画像の割合から大きく外れていない．この条件は，推論結果を学習データと比較可能にし，かつ検証可能にするためである．

推論を，692 枚の画像に対して行った．推論の結果を図 4 に示す．図 4 は、jpeg ファイルの名称が図の上部に表示される．画像は，不具合を推論した画像のサムネイルである．推論結果は，画像の下に表示される．この例では，“ Result : OK ” は不具合の可能性が低いと推論したことになる．一方，結果が “ Result : BUG ” と表示された場合，その画像の箇所は不具合が発生する可能性が高いと推論したことになる．Result の下の行は，BUG の確率と OK の確率を表している．この例では，BUG の確率は約 0.41、OK の確率は約 0.59 となる．

前述の通り，推論データは不具合を修正する前のプログラムの断片である．不具合の推論結果を評価するため，我々は不具合を解決したプログラムの修正内容を調査した．プログラムが修正されていない場合，プログラムのすべての画像のラベルは「OK」でなければならない．ま



(a) Epoch vs. Loss　　(b) Epoch vs. Accuracy

図 3. 学習経過

た，プログラムが修正されているのであれば，画像のラベルは修正箇所では「BUG」でなければならない．推論結果のラベルが，不具合を解決した結果のラベルと一致した場合，推論の結果は正しいと判断できる．

各プログラムは図 5 に X 軸の画像数と Y 軸の不具合数がプロットされている．画像の枚数はプログラムの大きさを示し，各プロットを原点から結ぶ線の傾きはプログラムの品質を示している．勾配が大きい場合，プログラムの品質は低くなる．

図には 2 つの階層データがある．プログラムを 3 つの層に分類した．層 1 に分類されたプログラムは、層 2 のプログラムよりも不具合が多く発生している．

推論結果を評価するため，我々は推論した不具合と実際に発見された不具合の傾向を検証した．検証した結果を，図 6 に示す．x 軸に画像数，y 軸に「BUG」の数を取る．

ラベル ”BUG”の結果に対応して，適合率と再現率および F 値を算出する．表 2 に結果を示す．

表 2 は左から, Pg. ソースコードに付与した連番, #Positive 推論処理で BUG と判断された画像の枚数，#True 不具合修正結果で BUG と判断された画像の枚数
#TP. 不具合修正結果と推論処理で一致した画像の枚数である．

しかし，CNN-BI システムで推論できない不具合があった．表 3 に推論できなかった不具合を示す．

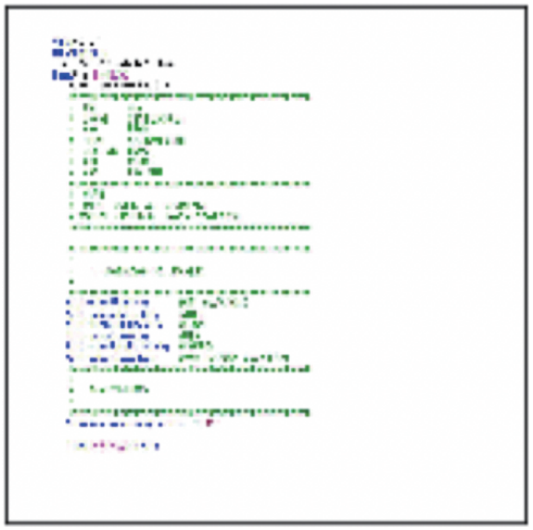
表 2. 適合率・再現率・F 値

Pg.	#Positive	#True	#TP.	Precision	Recall	F
a	8	9	6	0.75	0.67	0.71
b	14	3	2	0.14	0.67	0.24
c	28	15	7	0.25	0.47	0.33
d	18	13	3	0.17	0.23	0.19
e	91	59	41	0.45	0.69	0.55

SQL 文を含む画像は，障害が発生しやすいプログラムとして推論された．一般に，構文エラーはコンパイラーまたはコーディング・ツールによって検出されるが，実行前に SQL ステートメントのエラーを検出するのは困難である．CNN-BI システムは発見することが難しい不具合を推論することができた．しかし，CNN-BI システムによる推論では指摘が難しい不具合があった．

実際に発見された不具合と推論された不具合を比較したところ，実際に発見された不具合の次の画像に ”BUG” というラベルが付けられているケースが複数あった.「不具合と判定した画像の次画像に不具合の可能性があると推論していた」と仮定したときの適合率と再現率と F 値を，表 4 に示す．

learn006_000001.jpg



Result: OK

[[0.41111973 0.5888802]]

図 4. An example of the results of an inference.

取りこぼしが減少したため，再現率は増加した．しかし，適合率およびF値は減少した．

また、変数およびロジックが削除されたものと追加されたものは CNN-BI システムによる発見が困難であった.「ロジックの削除と追加以外の不具合を発見する」と仮定したときの適合率と再現率とF値を，表 5 に示す．推論修正と同様に取りこぼしが減少したため，再現率は増加した．適合率にはほぼ変化がなく、F値は再現率が増加したことに併せて増加している．変数およびロジックの削除と追加を推論しないことで，再現率が向上し，プログラムによっては，他の全ての不具合を推論できた．

推論の適合率を向上させるため，プログラムの構造に合わせて画像を作成した．プログラムの画像は，決まった行数で分けられているため，プログラムの構造を考慮していないことが原因になるのではないか考えたためである．プログラムの画像を作成する方法を，以下に示す．

1. 各プログラムを A5 横サイズで，最大 30 行のコードに分割した．
2. 関数・サブルーチン単位で切り分けを行った．
3. 30 行以上になる場合は，処理のインデントに合わせて 25～30 行の範囲で分割した．

上記の作成方法に従って画像を作成した．学習結果を

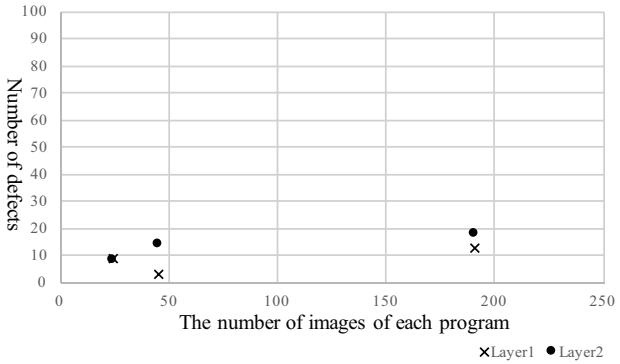


図 5. 各プログラムから作成した画像枚数とプログラムで実際に発見された不具合数の散布図.

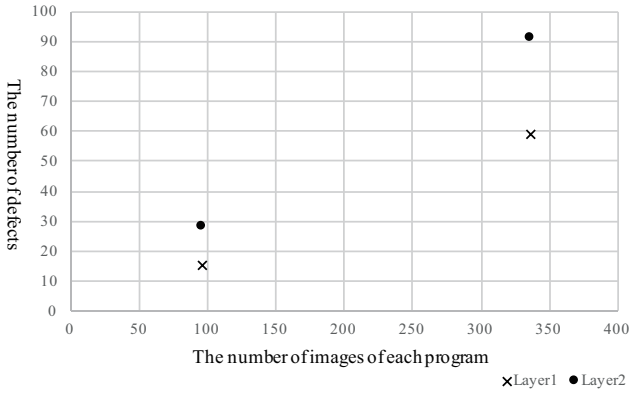


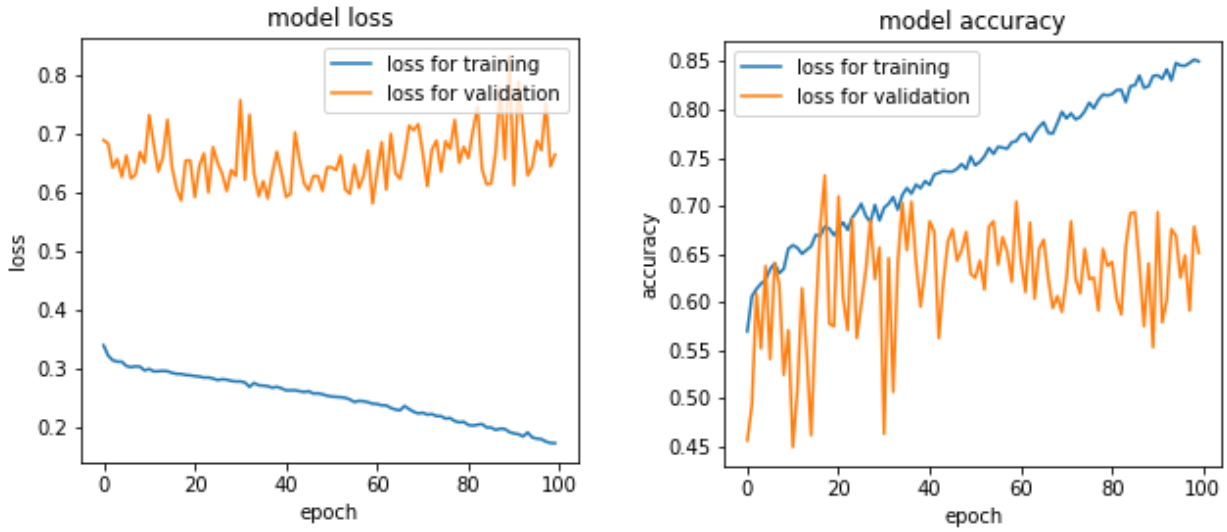
図 6. 画像数と推定された “ BUG ” の数の散布図

図 7 に示す．学習結果を確認したところ，学習の過程で検証データの損失及び精度が向上しない結果となった．これは，画像当たりのソースコード量が減ったため，特徴を捉えにくくなってしまったと考えられる．画像当りのソースコード量を減らさず，ソースコードの切り分けをする方法を考える必要がある．

4. 考察

4.1. プログラムの不具合推論は可能か

本稿では，不具合を起こしやすいプログラムを検出するため，プログラムの画像を CNN モデルに適用し，深層学習を行った．深層学習により，教師あり学習を用い



(a) Epoch vs. Loss

(b) Epoch vs. Accuracy

図 7. 画像作成方法変更後の学習経過

表 3. 推論できなかった不具合の内訳

Pg.	不具合の対応内容	件数
a	変数の削除	1
a	ロジックの削除	2
b	Ｉ Ｆ 文（条件の変数変更）の修正	1
c	呼出し先関数の修正	8
d	ロジックの追加	4
d	ロジックの修正	3
d	ロジックの削除	3
e	変数の追加・削除	13
e	ロジックの追加	1
e	ロジックの修正	2
e	呼出し先関数の修正	2

て不具合と画像の関係を学習した．不具合の学習を行った後，学習結果のモデルを使用して推論を行い，推論結果の検証を実施した．

4.1.1 推論結果

図 5 及び，図 6 は，各プログラムの画像数と実際の/推定された障害の数を用いた散布図を示している．x 軸は各プログラムのサイズを表し，y 軸はプログラムの不具合の発生しやすさを表す．散布図は，プログラムのサイズと不具合の発生しやすさとの間に，強い正の相関関係があることを示している．プログラムのサイズが大きくなるに従って障害の数が増加している．図に示すように，推論された不具合の数は実際の不具合の数よりも多かった．推論で慎重に検討する必要があるプログラムの箇所を指摘することにより，ソフトウェアの品質を向上させることができるため，問題無いと考えられる．

表 4. 推論修正後の適合率, 再現率及び F 値

Pg.	#Positive	#True	#TP.	Precision	Recall	F
a	14	9	7	0.50	0.78	0.61
b	22	3	2	0.09	0.67	0.16
c	51	15	12	0.24	0.80	0.36
d	31	13	4	0.13	0.31	0.18
e	153	59	50	0.33	0.85	0.47

4.1.2 不具合の推論結果

表 2 が示す通り，推定された不具合の再現率は最大 0.68 であった．しかし，表 3 の不具合より，推論処理に

表 5. 変数およびロジックの削除と追加を除いた場合の適合率, 再現率及び F 値

Pg.	#Positive	#True	#TP.	Precision	Recall	F
a	14	6	7	0.75	1	0.85
b	22	3	2	0.14	0.67	0.23
c	51	15	12	0.25	0.47	0.32
d	31	7	4	0.16	0.42	0.24
e	153	43	50	0.45	0.95	0.61

よって推論できなかった不具合を以下に示す.

- 条件文中の変数名が変更された.
- ロジックの追加及び削除.

CNN-BI システムが，上記の不具合を検出することは困難であった．これらの不具合は，画像に基づく推論では検出が難しいと考えられる．一方，SQL ステートメントを含む不具合は，検出ができていた．

我々の研究において最も重要な点は，我々がプログラムの品質を改善するため，この手法を適用する際，CNN-BI システムの弱点を明らかにすることである．我々は，CNN-BI システムによって，検出される不具合のあるプログラムの箇所と同様に，弱点を検討することが可能となる．

4.2. 推論の精度

適合率と再現率と F 値を，表 4 により示した．推論の結果，F 値が示している通り CNN-BI システムは高い適合率および再現率で推論することは期待できない．不具合の取りこぼしをしないため，適合率よりも高い再現率を得ることが重要である．CNN-BI システムによる推論は我々の手法によると，再現率は適合率よりも高い．適合率が再現率よりも高い場合，多くの不具合を見逃す可能性があるため，ソフトウェアの品質を向上させることは期待できない.

4.2.1 画像による不具合の推論

プログラムを画像にすることで，ソースコードの不具合を推測することは，SQL 文を文字列で記述しているプログラムの不具合を検出するのに効果的である．一方で，CNN-BI システムには弱点がある．変数やコードの追加や削除は検出が難しい.

CNN-BI システムの再現率は，すべての不具合を推論するとした場合，最大で 85%であった．我々は，不具合を起こしやすいプログラムの箇所を検出するため，画像による推論は可能であると結論付けることができた．しかし，プログラムの不具合が発生しやすい箇所を推論するための課題がある.

- CNN-BI システムの適合率の向上.

  適合率を向上させるためには，より多くの学習データが必要となる.

  また，画像の作成ルールを変更して学習を行った．関数及びサブルーチン単位で切り分けをしたが，画像 1 枚当たりの情報量を減らしてしまった．情報量が減ったことで，不具合の有る画像と無い画像で有意差も減ったために，特徴を捉えられなくなったと考えられる．画像の作成ルールは再検討が必要である．画像の作成方法を変更することには，CNN-BI システムによる推論の適合率を改善する可能性がある.

- CNN-BI システムの汎化能力の向上.
  本稿では，単一プロジェクトのプログラムソースコードを利用した．CNN-BI システムを他のプロジェクトに適用することは今後の課題である.

5. まとめ

本稿は，ソースコードを画像データに変換することによりプログラムの不具合を推論するため，学習済みデータセットを用いて転移学習を行った．我々のアプローチの鍵は，プログラムのソースコードから画像データを作成する事である．画像データから特徴を抽出するために CNN の長所を利用した.

我々は、以下の手順により，アプローチの有効性を実証した。

- 推論用プログラムに実際に見つかった不具合と比較して，推論結果を分析した.
- ソースコードの不具合について，推論結果を検証した.

検証結果より，完全ではないが，プログラムの不具合を推論することに成功した．本稿の問いに対する答えを以下に示す．

- CNN-BI システムは，プログラムの画像から不具合を起こしやすい特徴を抽出することができた．推論が完全ではないが，我々はシステムを使用して，重点的にプログラムレビューを行う箇所について指摘することができる．CNN-BI システムは，プログラムをレビューするため，どこに集中する必要があるか事前に決定するのに効果的である．
- プログラム中に存在する，いくつかの典型的な不具合の特定に成功した．また，問題のいくつかは，我々のアプローチで見つけるのが難しいことが分かった．
- 推論の精度は，我々の研究では許容可能であった．推論の精度を向上させるための課題を検討した．

本稿では、CNN を少数の学習データに適用した．推論の精度を向上させるため，学習データ（プログラム）を大量に用意し学習する必要がある．

今後，システム開発に於いて不具合の推論を行い，レビュー及びテストで特に注意してみるべき箇所を指摘することで，効率的な品質の向上に貢献したい．

参考文献

[1] B. Mayer. *Object-Oriented Software Construction, 2nd ed.* Prentice Hall, 2000.

[2] Karl Weiss, Taghi M. Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big Data*, 3(1):9, May 2016.

[3] K. Ogawa and T. Nakatani. Predicting fault proneness of programs with cnn. *11th International Conference on Agents and Artificial Inteligence*, 1:321–328, Feb 2019.

[4] Thomas J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, SE-2:308–320, 1976.

[5] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, June 1994.

[6] T. Zimmermann, R. Premraj, and A. Zeller. Predicting defects for eclipse. In *Third International Workshop on Predictor Models in Software Engineering (PROMISE'07: ICSE Workshops 2007)*, pages 9–15, May 2007.

[7] M. Lorenz and J. Kidd. *Object-Oriented Software Metrics*. Prentice-Hall, 1994.

[8] Quoc V. Le, Marc ' Aurelio Ranzato, Rajat Monga, Matthieu Devin, Kai Chen, Greg S. Corrado, Jeff Dean, and Andrew Y. Ng. Building high-level features using large scale unsupervised learning. In *Proc. of the 29th International Conference on Machine Learning, Edinburgh*, 6 2012.

[9] Masatoshi Morisaki. Deep learning use cases and their points and tips : 6. review source code with AI. *IPSJ Magazine*, 59(11):985–988, 10 2018.

[10] Masanari Kondo, Keita Mori, Osamu Mizuno, and Eun-Hye Choi. Just-in-time defect prediction applying deep learning to source code changes (in japanese). *Journal of Information Processing Systems*, 59(4):1250–1261, apr 2018.

[11] X. Yang, D. Lo, X. Xia, Y. Zhang, and J. Sun. Deep learning for just-in-time defect prediction. In *2015 IEEE International Conference on Software Quality, Reliability and Security*, pages 17–26, Aug 2015.

[12] Nathalie Japkowicz. Learning from imbalanced data sets: A comparison of various strategies. In *AAAI Technical Report WS-00-05*. AAAI, 2000.