卒 業 研 究 報 告 書

題　目　　画像分類による

ソフトウェア不具合予測システムの試作

指導教員　　水野　修　教授

京都工芸繊維大学 工芸科学部 情報工学課程

学生番号　14122046

氏　名　廣瀬　早都希

平成30年2月14日提出

画像分類によるソフトウェア不具合予測システムの試作

平成 30 年 2 月 14 日　　14122046　廣瀬　早都希

概　要

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含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・分類を行った．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．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．本研究では，学習データ数の少なさや，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の最適化等の問題点が多々あるため，一概に画像分類によるソフトウェア不具合予測が不可能とは言えないが，汎化性能が良くなる兆しが見られなかったため，ソフトウェアの変更点のテキスト画像によるソフトウェア不具合予測は非常に難しいという結果が得られた．

目　次

1. 緒言 **1**

2. 研究設問 **3**

3. 準備 **4**

3.1 Git 4

3.2 Commit Guru 4

3.3 Keras 5

3.4 TensorFlow 6

4. 実験準備 **7**

4.1 使用するソースコード 7

4.2 ソースコードの変更点の画像化 7

4.3 Commit Guru から不具合ラベルを取得 8

4.4 使用した機械学習モデルの構造 8

4.4.1 全結合ニューラルネットワーク 8

4.4.2 畳み込みニューラルネットワーク 14

4.5 評価尺度 16

5. 結果と考察 **19**

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？ 19

5.1.1 概要 19

5.1.2 結果と考察 20

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？ 20

5.2.1 概要 20

5.2.2 結果と考察 23

6. 妥当性の検証 **27**

6.1　概念的妥当性 27

6.2　外的妥当性 27

6.3　内的妥当性 27

7. 今後の課題 **29**

8. 結言 **32**

謝辞 **32**

参考文献 **33**

1. 緒言

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 (注 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 Xi.Yang ら [2]，S.Wang ら [3]，R.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．

そこで，本研究では，近藤らの研究と同様に，機械学習を分類機として利用し，

（注 1）: Commit Guru:http://commit.guru

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，ソースコードからテキスト情報を取り出すよりも，スクリーンショット等の手段により，手軽にソースコードの変更情報を取り出すことができると考えられるからである．

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 [6][7][8][9]．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．

以降の本報告書の構成を次に述べる．2 章では本研究の目的と検証すべき研究設問を提示する．3 章では本研究で必要となる事前知識を説明する．4 章では本研究で行なった事前準備について説明する．5 章ではそれぞれの研究設問の概要とそれに対する結果と考察を述べる．6 章では本研究の妥当性の検証を行う．7 章では今後の課題について述べる．8 章では本研究の結言を述べる．

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

本項目では，研究で用いた機能について説明する．

3.1 Git

Git (注 2) とは，Linux カーネル (kernel) のソースコード管理に用いるために Linus Benedict Torvalds によって開発された，代表的な分散型バージョン管理システムである．バージョン管理システムとは，特にソフトウェア開発でソースコードの管理に用いられる，コンピュータ上で作成・編集されるファイルの変更履歴を管理するためのシステムである．分散型バージョン管理システムはバージョン管理システムの一種で，複数のリポジトリ (ファイルやディレクトリの状態や変更履歴 (開発過程) が保存されるデータベース) を持つことができる．チーム開発から個人開発まで幅広く利用できるため，数多くのプロジェクトが Git で管理されている．

3.2 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 [9] で設定されているものと同様のものであり，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(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

(注 2) : Git:https://git-scm.com

経験数 (総 commit 数)) となっている．

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 [10] 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 non_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．

3.3 Keras

Keras (注 3) 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．迅速な実験が可能になったことで，研究のアイディアが出てから実験結果が出るまでの時間が短縮できるようになるため，より良い研究を行うことができる．次の場合で機械学習ライブラリを使用するならば，Keras を用いると良い．

- 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

（注 3）: Keras:https://keras.io/ja/

3.4 TensorFlow

TensorFlow（注 4）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 3.3 項で述べた Keras と TensorFlow を使用することとする．

（注 4）: TensorFlow:https://www.tensorflow.org

4. 実験準備

本項目では，実験前に行った準備について説明する．

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．

本来であれば，多種多様なプロジェクト (サーバーや WEB アプリ等) や異なるプログラミング言語で書かれているプロジェクトで実験を行うべきであるが，本研究ではソフトウェア不具合予測システムの試作ということで，特定のプロジェクトに注視して実験を行うこととする．

Git 上に公開されている commit 数は約 16000 個だが，今回は不具合の有無を Commit Guru によるラベル付けから得ているため，Commit Guru で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 4.1 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．不具合予測を行うことが目的であるので，削除によって不具合が発生するとは考えにくいことから，今回は追加・修正された箇所に焦点を当てて研究を行う．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 1028×1028 で RGB 画像として書き込む．なお，

フォントは“NotoMono-Regular”という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4.2に示す．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 に変更する．

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

本項目では，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．Keras(3.3項)とTensorflow(3.4項)を用いてモデルを作成した．

4.4.1 全結合ニューラルネットワーク

本研究ではまず全結合ニューラルネットワークモデルを作成し，それによる学習・分類を試みた．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．

```
@@ -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 4.1　git diff コマンドの出力例

```
                                   NodeConn,
from test_framework.util import (p2p_port, assert_equal)
    def send_blocks_until_disconnected(self, node):
            if not node.connection:
                node.send_message(msg_block(self.blocks[i]))
        node0 = BaseNode()
        connections = []
        connections.append(NodeConn('127.0.0.1', p2p_port(0), self.nodes[0], node0))
        node0.add_connection(connections[0])
        node0.wait_for_verack()
        node1 = BaseNode()  # connects to node1
        connections.append(NodeConn('127.0.0.1', p2p_port(1), self.nodes[1], node1))
        node1.add_connection(connections[1])
        node1.wait_for_verack()
        node2 = BaseNode()  # connects to node2
        connections.append(NodeConn('127.0.0.1', p2p_port(2), self.nodes[2], node2))
        node2.add_connection(connections[2])
        node2.wait_for_verack()
        node0.send_header_for_blocks(self.blocks[0:2000])
        node0.send_header_for_blocks(self.blocks[2000:])
        node1.send_header_for_blocks(self.blocks[0:2000])
        node1.send_header_for_blocks(self.blocks[2000:])
        node2.send_header_for_blocks(self.blocks[0:200])
        self.send_blocks_until_disconnected(node0)
            node1.send_message(msg_block(self.blocks[i]))
        node1.sync_with_ping(120)
        self.send_blocks_until_disconnected(node2)
        node0 = NodeConnCB()
        connections = []
        connections.append(NodeConn('127.0.0.1', p2p_port(0), self.nodes[0], node0))
        node0.add_connection(connections[0])
        node0.wait_for_verack()
        node0.send_and_ping(msg_block(block))
        node0.send_and_ping(msg_block(block))
        wait_until(lambda: "reject" in node0.last_message.keys(), lock=mininode_lock)
            assert_equal(node0.last_message["reject"].code, REJECT_OBSOLETE)
            assert_equal(node0.last_message["reject"].reason, b'bad-version(0x00000003)')
            assert_equal(node0.last_message["reject"].data, block.sha256)
            del node0.last_message["reject"]
        node0.send_and_ping(msg_tx(spendtx))
        node0.send_and_ping(msg_block(block))
        wait_until(lambda: "reject" in node0.last_message.keys(), lock=mininode_lock)
            assert node0.last_message["reject"].code in [REJECT_INVALID, REJECT_NONSTANDARD]
            assert_equal(node0.last_message["reject"].data, block.sha256)
            if node0.last_message["reject"].code == REJECT_INVALID:
                assert_equal(node0.last_message["reject"].reason, b'block-validation-failed')
                assert b'Negative locktime' in node0.last_message["reject"].reason
        node0.send_and_ping(msg_block(block))
        node0 = NodeConnCB()
        connections = []
        connections.append(NodeConn('127.0.0.1', p2p_port(0), self.nodes[0], node0))
        node0.add_connection(connections[0])
        node0.wait_for_verack()
        node0.send_and_ping(msg_block(block))
        node0.send_and_ping(msg_block(block))
        wait_until(lambda: "reject" in node0.last_message.keys(), lock=mininode_lock)
            assert_equal(node0.last_message["reject"].code, REJECT_OBSOLETE)
            assert_equal(node0.last_message["reject"].reason, b'bad-version(0x00000002)')
            assert_equal(node0.last_message["reject"].data, block.sha256)
            del node0.last_message["reject"]
        node0.send_and_ping(msg_tx(spendtx))
        node0.send_and_ping(msg_block(block))
        wait_until(lambda: "reject" in node0.last_message.keys(), lock=mininode_lock)
            assert node0.last_message["reject"].code in [REJECT_INVALID, REJECT_NONSTANDARD]
            assert_equal(node0.last_message["reject"].data, block.sha256)
            if node0.last_message["reject"].code == REJECT_INVALID:
                assert_equal(node0.last_message["reject"].reason, b'block-validation-failed')
                assert b'Non-canonical DER signature' in node0.last_message["reject"].reason
        node0.send_and_ping(msg_block(block))
    NodeConn,
    p2p_port,
        node0 = BaseNode()
        connections = []
        connections.append(NodeConn('127.0.0.1', p2p_port(0), self.nodes[0], node0))
        node0.add_connection(connections[0])
        node0.wait_for_verack()
            node0.send_message(block_message)
        node2 = BaseNode()
        connections.append(NodeConn('127.0.0.1', p2p_port(2), self.nodes[2], node2))
        node2.add_connection(connections[1])
        node2.wait_for_verack()
        node2.send_message(getdata_request)
        wait_until(lambda: sorted(blocks) == sorted(list(node2.block_receive_map.keys())), timeout=5, lock=mininode_lock)
            for block in node2.block_receive_map.values():
        # test_nodes[0] will only request old blocks
        # test_nodes[1] will only request new blocks
        # test_nodes[2] will test resetting the counters
        test_nodes = []
        connections = []
            test_nodes.append(TestNode())
            connections.append(NodeConn('127.0.0.1', p2p_port(0), self.nodes[0], test_nodes[i]))
            test_nodes[i].add_connection(connections[i])
        [x.wait_for_verack() for x in test_nodes]
```

図 **4.2**　変更点の画像化例

本研究で使用したのは全結合ニューラルネットワークである．全結合ニューラルネットワークモデルの概要は図 4.3 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 項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 4.1 図 4.4) 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 4.2 図 4.5) を用いる．

$$Relu(x) = \begin{cases} x & (x \geq 0) \\ 0 & (x < 0) \end{cases} \tag{4.1}$$

$$softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \quad (i = 1, 2, \cdots, n) \tag{4.2}$$

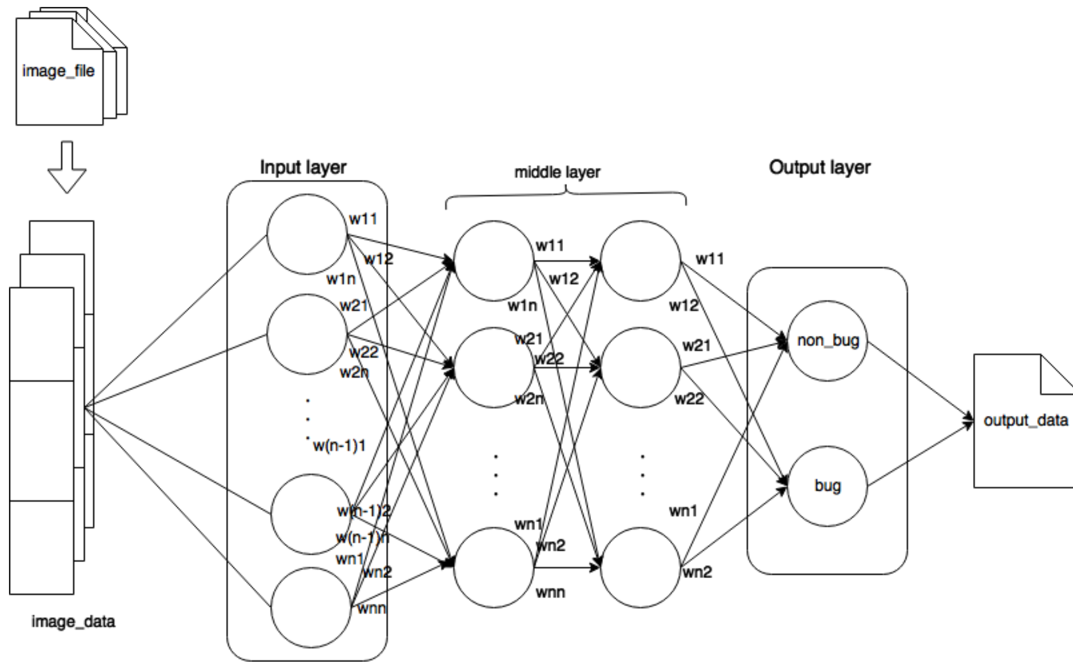

図 4.3　全結合ニューラルネットワークモデルの概要

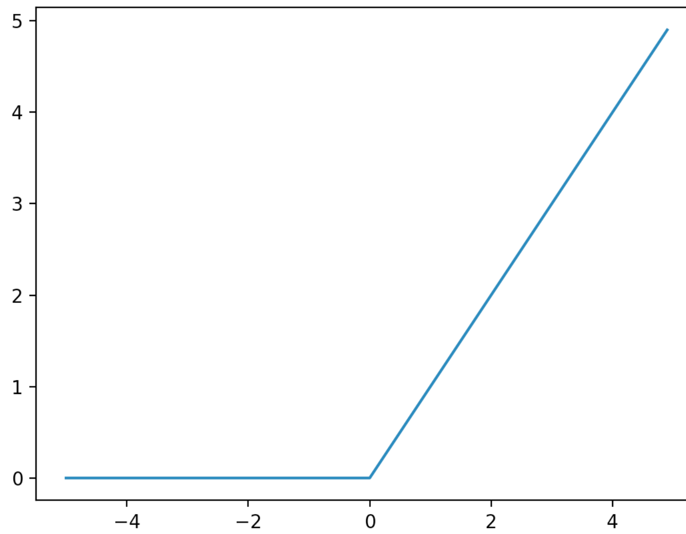

図 4.4　ReLU 関数

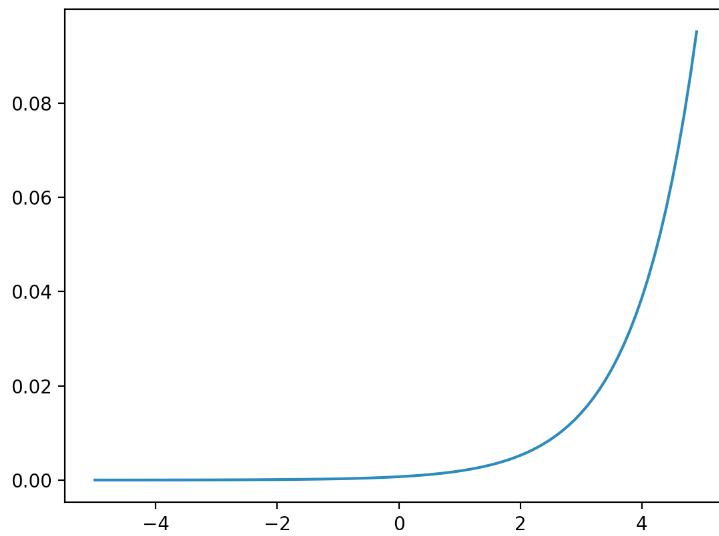

図 4.5　softmax 関数

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 4.6 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様な処理を行う．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．

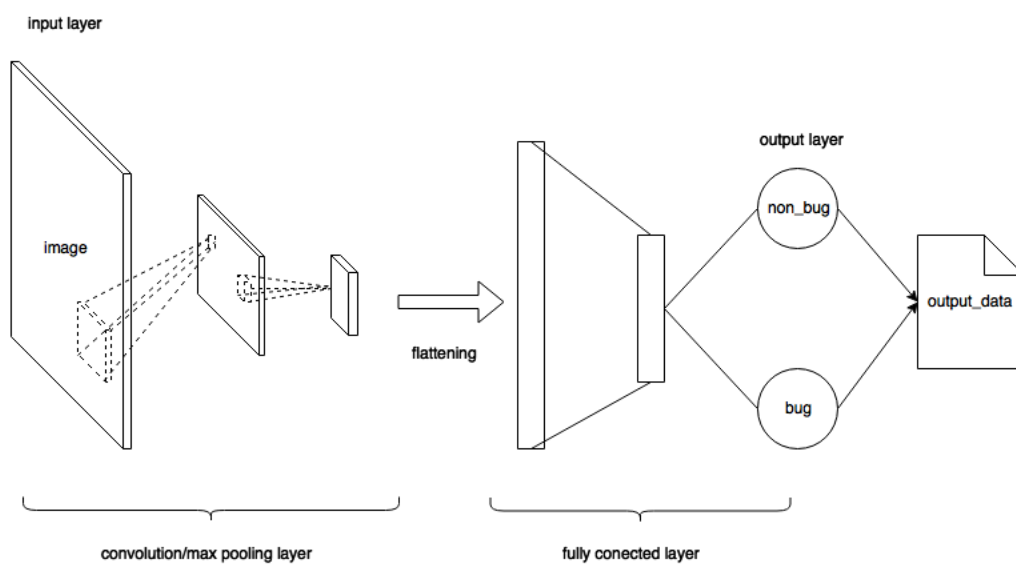

図 **4.6**　畳み込みニューラルネットワークモデルの概要

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 4.1 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．

(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．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 4.1) を用いて定義すると以下のようになる．

$$Recall = \frac{TP}{TP + FN} \tag{4.3}$$

(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 4.1) を用いて定義すると以下のようになる．

表 4.1　混合行列

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

$$Precision = \frac{TP}{TP + FP} \tag{4.4}$$

(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．

$$F1 = \frac{2 \times Recall \times Precision}{Recall + Precision} \tag{4.5}$$

(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 4.1) を用いて定義すると以下のようになる．

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{4.6}$$

5. 結果と考察

この章では，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．

ニューラルネットワークにおいて実験データに偏り (例えば，学習データやテストデータに不具合のあるものが 1 つも含まれてない等) があると正しく実験を行うことができない．そこで，本研究では，実験データの偏りを減らすため，10 重交差検証を行うこととした．10 重交差検証とは，使用する実験データを 10 個のグループに分割し，そのうちの 9 個のグループを学習データ，残った 1 個のグループをテストデータとし，実験を行う．これを 10 個のグループそれぞれがテストデータとなるまで繰り返すという手法である．今回，使用する実験データの総数が約 15000 個なので，学習データは約 14000 個，テストデータは約 1500 個となる．しかし，想定以上に畳み込みニューラルネットワークの学習に時間がかかってしまい，期間内での実験が不可能であったため，学習データ約 14000 個，テストデータ約 1500 個の 1 組だけを用いて実験を行うこととする．(つまり実験回数は 1 回である．) ただし，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 5.1 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 5.1 と図 5.2 に示す．

図 5.1 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．

図 5.2 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．

ソフトウェア不具合予測において，重要となってくるのが，不具合検出の精度だと考えられる．例えソフトウェア不具合予測が手軽にできたとしても，不具合検出の精度が低ければ，ソフトウェアの品質を保つことには繋がらないと考えられるからである．

そこで，RQ1 で学習が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，画像分類を行い，機械学習の精度評価でよく用いられている評価尺度を用いて，本実験で用いた畳み込みニューラルネットワークの評価を行うこととする．

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．

表 **5.1**　学習データ・テストデータに含まれる画像データの内訳

	number of files
train-bug	2071
train-nonbug	11984
test-bug	208
test-nonbug	1354

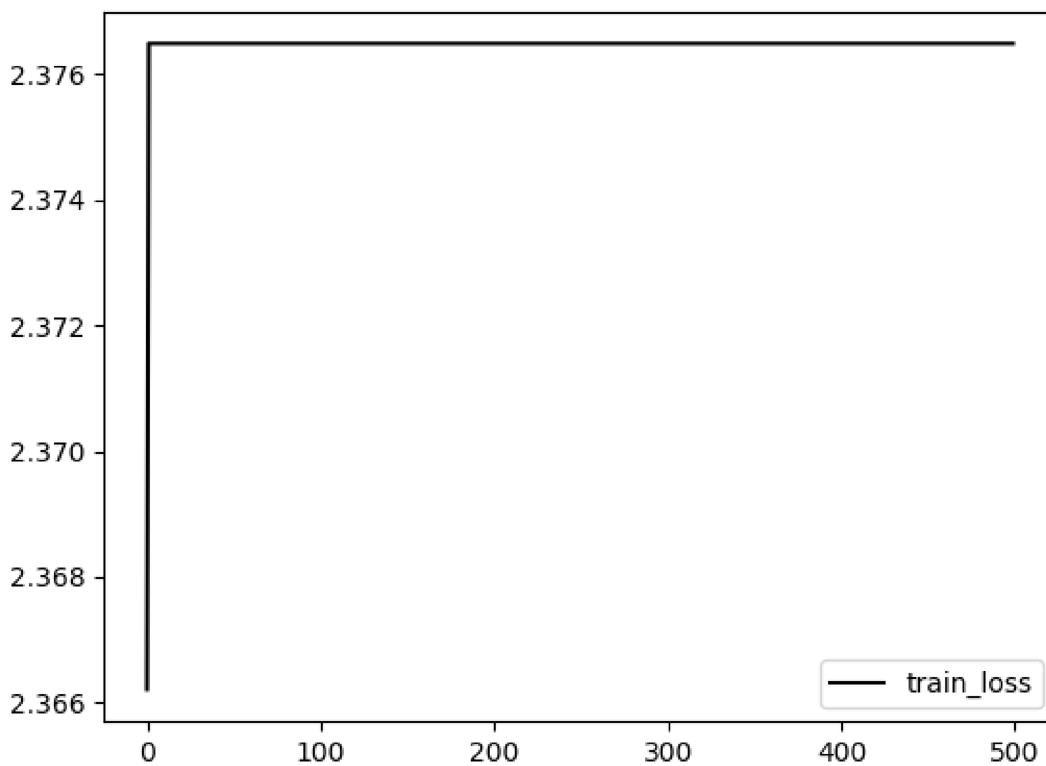

図 **5.1**　全結合ニューラルネットワークの学習時の訓練誤差 **(train_loss)**

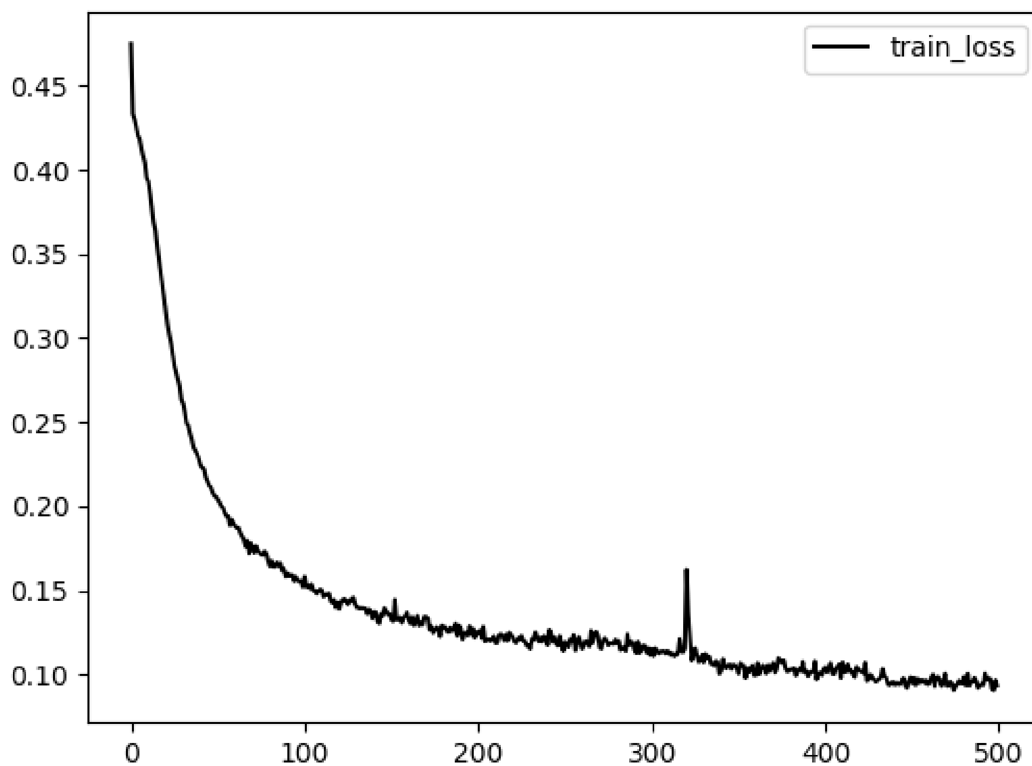

図 5.2　畳み込みニューラルネットワークの学習時の訓練誤差 (train_loss)

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 5.2 に示す．表 5.2 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 5.3 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 5.3 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．

表 **5.2**　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 **5.3**　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

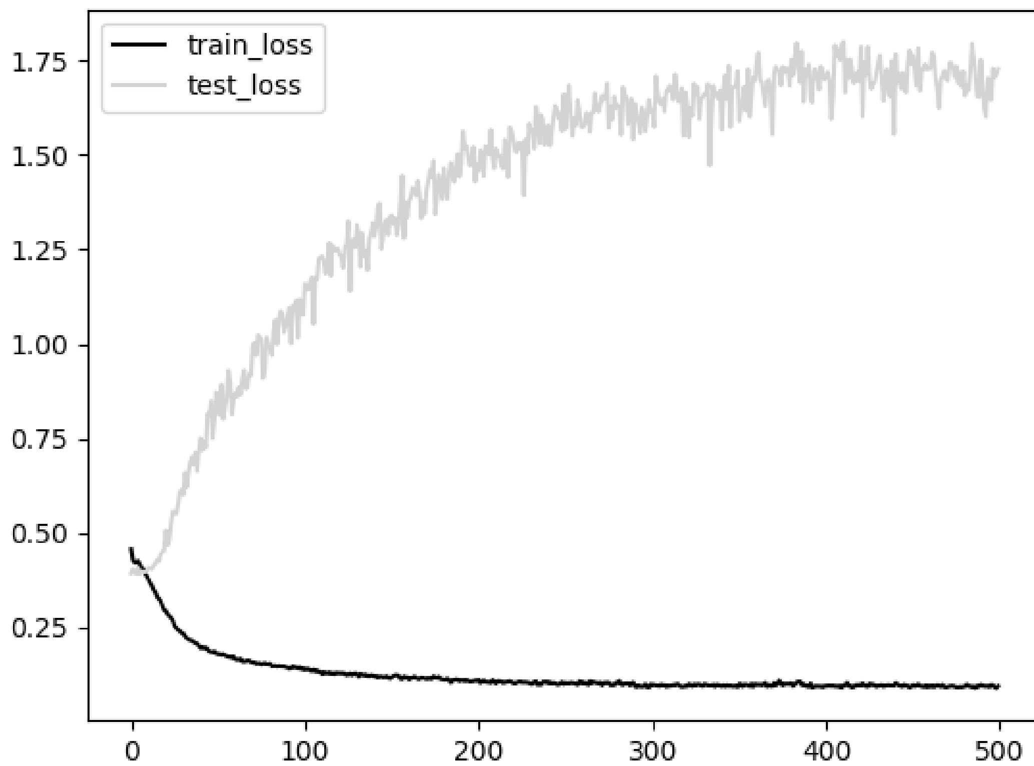

図 **5.3**　畳み込みニューラルネットワークの訓練誤差とテスト誤差

6. 妥当性の検証

この章では妥当性の検証を行う．

6.1 概念的妥当性

概念的妥当性では，本実験で用いた評価手法が適切であったかの検証を行う．

本実験で用いたのは，機械学習後の画像分類における精度評価として recall(再現率)，Precision(適合率)，F1-score(F1 値)，accuracy(適合率) である．これらは，機械学習における評価指標として一般的に用いられている評価手法であるので，本実験での評価手法として妥当なものであると考えられる．

しかし，本実験では，1 つの学習データ・テストデータのみを使用しており，また，不具合がある画像がとても少ないという偏りのあるデータを用いた実験であった．そのため，今回使用した学習データが本実験に適した分散を持っているとは言えない．

よって，概念的妥当性は保障されていないと考えられる．概念的妥当性を高めるためには，より多くのデータを用意し，データの偏りを取り除く必要がある．

6.2 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検証する．

本実験では，C++で書かれた 1 つのプロジェクト“bitcoin”のみを実験データとして利用した．そのため，その他の言語で書かれたプロジェクトに対しても本結果は適応可能かの判断はできない．

よって，本実験の一般性は示されない．実験対象とするプロジェクト数の増加やソースコードに使用されている言語数の増加によって，外的妥当性を高める必要がある。

6.3 内的妥当性

内的妥当性では，本実験の手法の妥当性について検証する。

本実験で利用した変更点に含まれる不具合有無のデータはCommit Guruで公開されているデータである．つまり，使用したプロジェクトの変更点に不具合が含まれているかの判定は，Commit Guruのラベル付けに依存しているということになる．しかし，Commit Guruの不具合ラベル付けは完全ではない．例えば，Commit Guruで用いられているラベル付け手法はコミットメッセージを利用したものである．コミットメッセージに特定のキーワードが含まれているか否かによって不具合の判定を行なっている．つまり，コミットメッセージに特定のキーワードが含まれていないコミットに不具合が含まれていた場合，見逃す場合がある．Commit Guruでの不具合ラベル付けが正しくない場合，本実験の手法の内的妥当性は低いと考えられる．

また，本実験で作成したプログラムに対しての妥当性も疑われる．作成したプログラム内に不具合が存在していた場合もあるため，内的妥当性は低いと考えられる．

7. 今後の課題

本研究の結果・考察 (5 章) 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．
- 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．

各項目について説明を行う．

1. 今回の実験では，画像分類の中でも複雑なテキスト画像の分類という難しい学習を必要とするものであった．そのため，畳み込みニューラルネットワークの中間層の値も自然と大きくなった．しかし，今回与えた学習データ数では中間層で全ての学習を行うのに少なすぎたため，過学習となった可能性が高い．よって，実験で使用するデータ数を増やすことで畳み込みニューラルネットワークの中間層の学習を終えられるようにすると，正しく画像分類が行われるのではないかと考えられる．
2. 1 つ目の課題でも言及しているように，本研究の畳み込みニューラルネットワークは中間層の値が大きくなりがちである．よって，この中間層の値を適切に設定することができれば，畳み込みニューラルネットワークが過学習せずに

正しく学習が行われるのではないかと考えられる．本研究では，中間層の値を適切に設定するための検証を行っていないため，適度な値を与えていた．そのため，中間層の適切な値の調整を行うことを今後の課題としてあげる．

また，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. 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．

各項目について説明を行う．

1. 先に挙げた課題 1 とこの課題は概念的妥当性に対する課題である．概念的妥当性を満たすためには，実験に使用するデータの分散が実験に適したものとなっていることが必要である．本研究では，1 種類の学習データとテストデータを用いて 1 度の実験のみ行っただけである．使用したデータに含まれる不具合有無の割合も，明らかに不具合有りが少なく，偏りのあるデータといえる．そのためデータ数を増やすことで，データの偏りを減らすことを課題として挙げる．
   しかし，データ数を増やすことのみがデータの偏りを減らす方法ではない．本研究では実験期間の都合上，1 度の実験のみを行っただけである．当初の予定では，10 重交差検証という方法を取り，実験データの偏りを減らすことを考えていた．よって，今後は 10 重交差検証を用いて，よりデータの偏りを減らすことを課題とする．
2. この課題は外的妥当性に対する課題である．本研究では，1 つのプロジェクト，1 つのプログラミング言語に対してのみ実験を行なった．そのため，外的妥当性が低いと考えられる．よって，多様なプロジェクト (サーバー，Web アプリケーション，開発ツール等) に対して，この研究が適用可能かを検証する必要がある．また，ソースコードで使用されている言語も 1 種類のみではなく，

複数の汎用プログラミング言語に対して検証を行うべきである．よって，課題の 1 つとして，実験に使用するプロジェクト数の増加を挙げるものとする．

3. 内的妥当性に対する課題としてこれを挙げる．作成したプログラムに不具合が潜む可能性を 100%否定することはできない．よって，ダブルチェックを行ったり，簡易なテストを行って，プログラム内に不具合があるかを検証する必要がある．

以上を今後の課題として挙げる．これらを検証することで，畳み込みニューラルネットワークの画像分類によるソフトウェア不具合予測システムについて再度検討を行う．

8. 結言

ソフトウェア不具合予測に関しては，様々な研究がなされている．近年では，ソフトウェア不具合予測の性能を高めるために，機械学習が行われている例もあるが，画像分類を用いてソフトウェアの不具合を分類する研究は行われていない．よって，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われないかを再度検証する．

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，本報告書の作成に至るまで，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．本報告書執筆にあたり貴重な助言を多数頂きました，ソフトウェア工学研究室　黒田翔太 先輩，田中健太郎 先輩，中川要 先輩，西浦生成 先輩，原田禎之 先輩，小林勇揮 先輩，近藤将成 先輩，洪浚通 先輩，植村佳治 君，北村紗也加 さん，中村勝一 君，渡辺大輝 君，インタラクティブ知能研究室　廣田敦士 先輩，森秀晃 先輩，小野理彩子 さんをはじめとする，本学情報工学専攻・情報工学課程の皆様，学生生活を通じて著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] C. Rosen，B. Grawi，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, Sept. 2015.

[2] X. Yang，D. Lo，X. Xia，Y. Zhang，J. Sun，“Deep learning for just-in-time defect prediction,” QRS ’15 Proceedings 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，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，P. Kakkar，“Software module fault prediction using convolutional neural network with feature selection,” International Journal 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] S. Kim，J. E. James Whitehead，Y. Zhang，“Classifying software changes: Clean or buggy?,” IEEE Transactions on Software Engineering, vol.34, no.2, pp.181–196, March 2008.

[7] L. Aversano，L. Cerulo，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, ew York, NY, USA, Sept. 2007.

[8] T. Jiang，L. Tan，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.

[9] Y. Kamei, E. Shihab, B. Adams, A.E. Hassan, A. Mockus, A. Sinha, N. Ubayashi, "A large-scale empirical study of just-in-time quality assurance,".

[10] A. Hindle, D.M. German, 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.