THE IEICE TRANSACTIONS ON INFORMATION AND SYSTEMS (JAPANESE EDITION)

電子情報通信学会 論文誌 D 情報・システム

VOL. J104-D NO. 8
AUGUST 2021

情報・システムソサイエティ

一般社団法人 電子情報通信学会
THE INFORMATION AND SYSTEMS SOCIETY
THE INSTITUTE OF ELECTRONICS, INFORMATION AND COMMUNICATION ENGINEERS

論　文

深層学習を用いたソースコード分類手法の比較調査

藤原　裕士[†a)]　　崔　　恩瀞[††]　　吉田　則裕[†††]　　井上　克郎[†]

Comparing Deep Learning-based Approaches for Source Code Classification

Yuji FUJIWARA[†a)], Eunjong CHOI[††], Norihiro YOSHIDA[†††], and Katsuro INOUE[†]

あらまし　近年，深層学習を用いてソースコード分類を行う種々の手法が提案されている．これらの手法では，ニューラルネットワークにソースコードのトークン列などを学習させることでソースコード分類を行う．そのとき，ソースコード分類に有効でないニューラルネットワークやソースコード表現を学習に利用すると学習効率が低下するため，適切なニューラルネットワークやソースコード表現を選定する必要がある．しかし，どのニューラルネットワークやソースコード表現の組合せが高精度なソースコード分類手法の実現に有効かは明らかになっていない．本研究では，深層学習を用いたソースコード分類手法の比較調査を行う．まず，既存研究で広く利用されているニューラルネットワークを三つ選択した．次に，ニューラルネットワークにソースコードのトークン列または抽象構文木を学習させた計 6 種類のソースコード分類手法の精度を比較した．その結果，ソースコードのトークン列を学習した再帰型ニューラルネットワークの精度が最も高いことを確認した．また，深層学習と非深層学習の手法のソースコード分類精度を比較し，深層学習の手法の分類精度が高いことを確認した．

キーワード　深層学習，ソースコード分類，順伝播型ニューラルネットワーク，再帰型ニューラルネットワーク，グラフ畳み込みネットワーク

1. まえがき

ソフトウェア開発を効率良く行うために，開発者は既存のソースコードを頻繁に再利用する [1], [2]．ソースコード分類手法は，あらかじめ用意されたクラスに基づいて，入力として与えられたソースコードがどのクラスに属する既存ソースコードと類似しているかを自動で識別する手法である．開発者は，このソースコード分類手法を用いることで，再利用対象のソースコードを効率的に特定することができる．

現在まで多くのソースコード分類手法が提案されてきた [3]〜[9]．特に近年では深層学習を用いてソースコード分類を行う手法が提案され，高い分類精度を示した [6], [7]．これらの既存手法は様々なニューラルネットワークにソースコード表現を学習させることで実現されている．しかし，ニューラルネットワークやソースコード表現がどのようにソースコード分類の精度に影響し，どのニューラルネットワークやソースコード表現を利用することで高精度なソースコード分類の実現ができるかは明らかになっていない．また，深層学習を用いた既存のソースコード分類モデルは複雑な構造であるため，どのニューラルネットワークやソースコード表現が高精度なソースコード分類に有効かを把握することが難しい．

そこで本研究では，高精度なソースコード分類に有効なニューラルネットワークとソースコード表現の組合せについて調査するために，深層学習を用いたソースコード分類手法の精度を比較する．本研究を行うにあたり，リサーチクエスチョン（以降，RQ）を設定した．

RQ　高精度なソースコード分類を実現できるニューラルネットワークとソースコード表現の組合せは何か

この RQ に回答するためにまず，既存のソースコード分類手法でよく用いられるニューラルネットワークである，順伝播型ニューラルネットワーク，再帰型

† 大阪大学，吹田市
Osaka University, 1–1 Yamadaoka, Suita-shi, 565–0871 Japan
†† 京都工芸繊維大学，京都市
Kyoto Institute of Technology, Matsugasakihashikami-cho, Sakyo-ku, Kyoto-shi, 606–8585 Japan
††† 名古屋大学，名古屋市
Nagoya University, Furo-cho, Chikusa-ku, Nagoya-shi, 464–8601 Japan
a) E-mail: y-fujiwr@ist.osaka-u.ac.jp
DOI:10.14923/transinfj.2020JDP7068

ニューラルネットワーク，グラフ畳み込みネットワークの三つを選択した．次に，選択したニューラルネットワークにソースコードのトークン列または抽象構文木（Abstract Syntax Tree，以降，AST）を学習させ，計6種類のソースコード分類手法の精度を比較した．

その結果，再帰型ニューラルネットワークにソースコードのトークン列を学習させた手法の分類精度が最も高いことが分かった．

以上の調査によって，深層学習を用いたソースコード分類手法の中で最も分類精度が高い手法が明らかになった．しかし，深層学習を利用せずに高精度なソースコード分類が実行できる可能性がある．この前提を確認するために，深層学習を用いるソースコード分類手法と深層学習を用いないソースコード分類手法の精度の比較を行った．その結果，深層学習を用いる手法の方が分類精度が高く，深層学習はソースコード分類に有効であることが明らかになった．

本研究の貢献は以下の点である．

- 既存研究で広く利用されている3種類のニューラルネットワークに対してソースコードのトークン列またはASTを学習させ，計6種類のソースコード分類手法の精度比較を行うことで，どのソースコード分類手法の精度が高いかを調査した．その結果，再帰型ニューラルネットワークにソースコードのトークン列を学習させる手法の分類精度が最も高いことが分かった．
- 深層学習を用いたソースコード分類手法と用いない手法の精度比較を行った．その結果，深層学習を用いたソースコード分類手法の方が精度が高く，深層学習はソースコード分類に有効であることが分かった．

以降，**2.** では，比較調査の背景として，ソースコード分類と，ソースコード分類に用いられる代表的なニューラルネットワークについて述べる．**3.** では，深層学習を用いたソースコード分類手法の比較調査の説明と結果の考察を行い，高精度なソースコード分類を実現することができるニューラルネットワークとソースコード表現の組合せについて述べる．**4.** では，深層学習を用いる手法と用いない手法の分類精度の比較を行い，深層学習を用いる手法が高い精度を示すか確認する．**5.** では，本研究の妥当性の脅威について述べる．**6.** では，関連研究として，深層学習を用いてソースコード解析を行う既存研究について述べる．最後に，**7.** でまとめと今後の課題について述べる．

2. 背　　景

2.1 ソースコード分類

本研究におけるソースコード分類手法とは，既存ソースコードを構文的及び意味的類似ソースコードごとに分割した n 個の クラス $C_1 \ldots C_n$ に対して，入力として与えられたソースコードを，入力ソースコードの構文的及び意味的類似ソースコードが含まれるクラス $C_i (1 \leq i \leq n)$ に自動で分類する手法である．この手法を用いることでソフトウェアを効率的に開発することができる．例えば，ソースコードを自動で機能ごとに分類することで，大規模なソフトウェアリポジトリに新たに登録されたソースコードに対して機能に関するタグを自動で付与することができる．開発者はこのタグを用いることで，必要な機能をもった既存ソースコードの再利用を容易に行うことができる．このようにソースコード分類手法を活用することで，ソフトウェア開発の生産性の向上が期待できる．

ソースコード分類手法は類似ソースコード検索に応用することができる．まず，検索対象のソースコードを類似ソースコードごとのクラスに分割し，検索クエリのソースコードに対してソースコード分類を行う．そして，分類されたクラスに含まれるソースコードに，検索クエリソースコードとの類似度に応じたランキングを付与し，このランキングに従って，分類されたクラスに含まれるソースコードを検索結果として出力する．また，同じクラスに分類されたソースコードをコードクローン（ソースコード中に存在する互いに一致または類似部分をもつコード片）として検出することで，ソースコード分類手法をコードクローン検出に応用することができる．

ソースコード分類に関する研究では，現在まで，記述言語による分類 [3]，コンポーネント間の依存関係による分類 [4]，プログラムの意味（機能性）による分類など，多様な手法が提案されている．また，プログラムの意味によるソースコード分類は様々な粒度で取り組まれており，ソフトウェア単位の分類手法 [5] や，メソッド単位の分類手法 [6]〜[9] などが存在する．近年，深層学習を用いることで高い精度でソースコード分類を行う手法が提案されている [6], [7]．これらの手法で作成される深層学習モデルは，入力ソースコードに対してクラスごとに分類確率を計算し，最も確率が高いクラスを出力する．

2.2 ソースコード分類に用いられる代表的なニューラルネットワーク

一般的に深層学習とは，多くの層を利用し非線形情報の処理を行うことで，対象の特徴抽出や特徴変換，パターン分析，分類などのタスクを解決する機械学習の手法の一つである [10]．ニューラルネットワークの場合，1 層以上の隠れ層を利用することで非線形情報の処理を行うことが可能である [11]．そのため，本研究では，入力層と出力層に加えて 1 層以上の隠れ層を利用するニューラルネットワークを深層学習の手法と定義する．以降，本節ではソースコードの分類に用いられる代表的なニューラルネットワークについて説明する．

2.2.1 順伝播型ニューラルネットワーク

順伝播型ニューラルネットワーク（Feedforward Neural Networks，以降，FNN）[12] とは，ネットワークにループ構造が含まれない標準的なニューラルネットワークである．最初期は入力層と出力層のみからなる機械学習の手法だったが，隠れ層を増やすことで線形分離不可能な問題を解くことができる深層学習の手法として研究に利用されている．このニューラルネットワークは，ニューロンと呼ばれる，単純なベクトル計算を行うための要素を接続することで構成されている．ソースコードのメトリクスを並べるなどの方法でソースコードをベクトル化することで，FNN をソースコード分類に利用することができる [13]〜[15]

図 1 は 8 個のニューロン n_{11}〜n_{32} で構成される，3 層の FNN の例である．FNN の入力と出力はともにベクトルであり，そのベクトルの次元はネットワーク構造に依存する．図 1 の例では入力が 3 次元，出力が 2 次元のベクトルである．また，ネットワークの機能は，各ニューロン間の接続に設定されている重みとバイアスと呼ばれる，学習によって調整される値に依存する．FNN の学習では，入力ベクトルと出力ベクトルの組を FNN に与え，入力ベクトルを FNN に与えた際に対応する出力ベクトルが出力されるように FNN の内部パラメータを調整する．これにより，FNN は入力ベクトルと出力ベクトルを対応づけることが可能になる．

ソースコード分類の既存研究 [6] で比較対象に選択され，高い分類精度を示していることから，本研究でも FNN を比較対象に選択した．

2.2.2 再帰型ニューラルネットワーク

再帰型ニューラルネットワーク（Recurrent Neural Networks，以降，RNN）[12] とは，ベクトルの系列が

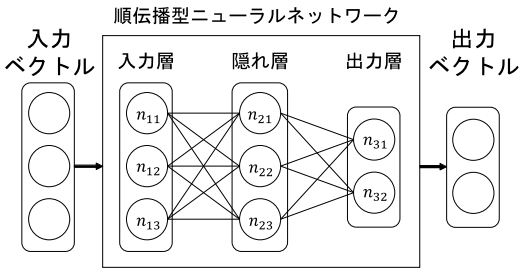


図 1　順伝播型ニューラルネットワークの例

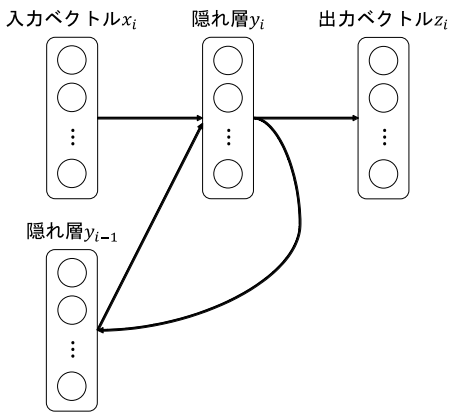


図 2　再帰型ニューラルネットワークの例

入力として与えられ，入力ベクトルの値だけでなく，入力ベクトルの順番にも出力が影響されるニューラルネットワークである．ソースコードはトークン列などの系列で表現することが可能なため，RNN はソースコード分類に用いられている [7], [16], [17].

RNN の例を図 2 に示す．この図から分かるように，RNN は FNN と違いネットワークにループ構造が含まれている．RNN では入力ベクトル系列に含まれるベクトルが一つずつ順に入力されるたびに計算が行われる．RNN における i 回目 $(1 \leq i \leq n)$ の計算では，i 番目のベクトル x_i と同時に，$i-1$ 番目のベクトルを入力した後のニューラルネットワークの隠れ層 y_{i-1} が RNN に入力される．この二つの入力を用いて RNN の隠れ層 y_i と出力 z_i の計算が行われる．このような手順で計算が行われるため，1〜i 番目の全ての入力ベクトルの値や入力順序が RNN の出力に影響する．

また，代表的な RNN の一つに，LSTM（長・短期記憶再帰型ニューラルネットワーク，Long-Short Term Memory recurrent neural network）[18] がある．LSTM は RNN の隠れ層を LSTM block に置き換えることによって，一般的な RNN と比べて更に長期的な依存関

係の学習が可能になったニューラルネットワークである．ソースコード分類の既存研究 [7] で比較対象に選択され，高い分類精度を記録しているため，本研究でも LSTM を比較対象に選択した．

2.2.3 グラフ畳み込みネットワーク

グラフ畳み込みネットワーク（Graph Convolution Networks，以降，GCN）[19] とは，グラフの隣接ノードを畳み込んでいくことによってノードやエッジ，グラフ全体の特徴を抽出するニューラルネットワークである．ソースコードは AST などのグラフで表現することができるため，GCN はソースコードの分類に利用されている [20]．GCN はグラフを学習可能なニューラルネットワークの中でも比較的新しいニューラルネットワークである．GCN が提案される前にまず，GCN の前身となる技術として，グラフニューラルネットワーク（Graph Neural Networks，以降，GNN）[21] が提案された．GNN は，グラフ構造に対して深層学習を行うために開発されたニューラルネットワークである．その後，画像認識の分野で畳み込みを利用したニューラルネットワークが高い精度を示していることから，グラフにも畳み込みを適用することで精度が高くなると考えられ，GCN が提案された．

深層学習を用いてソースコード分類を行う既存研究 [6], [7], [17] では，グラフの学習を行う際，ニューラルネットワークの入力形式に合わせて元のグラフを変形する．しかし，GCN ではグラフの変形は不要であるため，グラフの構造情報が欠落しないという利点がある．したがって GCN を用いることで，グラフを変形する必要があるニューラルネットワークに比べ，グラフに含まれる情報をより正確に利用した学習を行うことができる．GCN の畳み込み層の例を図 3 に示す．図 3 は，右上のグラフの中央のノード 0 のベクトル表現を計算する手順について説明している．ノード 0 の畳み込み $n+1$ 層目におけるベクトルは，隣接ノードの n 層目におけるベクトルと他ノードからのエッジ（ingoing，outgoing），ノード 0 にループするエッジ（self-loop）の重みから中間ベクトルを計算し，エッジごとの中間ベクトルを全て足し合わせたベクトルを ReLU などの活性化関数（ネットワークの出力を補正する関数）に入力することで得られる．このように GCN では，ノード 0 のベクトル表現及びノード 0 に隣接しているノード 1，2，3 のベクトル表現に基づいて，ノード 0 のベクトル表現が計算される．

グラフの学習が可能なニューラルネットワーク

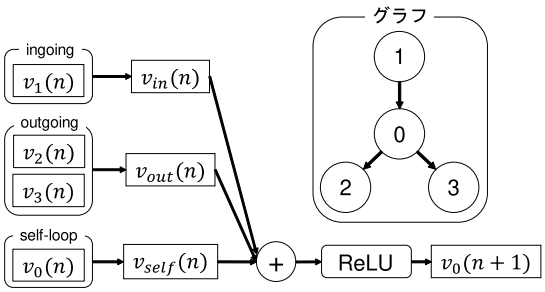


図 3　GCN の畳み込み層の例

を利用したソースコード分類に関する既存研究 [6], [7], [17], [20] は近年増加しているため，分類精度の調査を行う必要がある．また，GCN はグラフに適用するニューラルネットワークの中で比較的新しいため，高い精度の分類が期待できる．よって，本研究では GCN を比較対象に選択した．

3. 深層学習を用いたソースコード分類手法の精度比較調査

近年，深層学習を用いてソースコード分類を行う種々の手法が提案されている [6], [7]．これらの手法における深層学習モデルは複雑な構造をしており，様々なニューラルネットワークやソースコード表現を利用する．しかし，これらのニューラルネットワークやソースコード表現がどの程度ソースコード分類の精度に影響を及ぼすかは明らかになっていない．そこで本研究では，既存研究で広く利用されているニューラルネットワークとソースコード表現を用いたソースコード分類手法の精度の比較を行う．この比較を行うことで，高い精度でソースコード分類ができるニューラルネットワークとソースコード表現の組合せについて調査する．本研究では以上の調査を行うにあたって，**1.** で述べた RQ を設定した．

3.1 ベンチマーク

本研究では，ソースコード分類のベンチマークとして，BigCloneBench [22] を用いた．BigCloneBench とは，Java で開発されたオープンソースソフトウェアからメソッド単位でソースコードを収集したベンチマークである．BigCloneBench の開発者らによって収集されたメソッドは，そのメソッドが実現する機能に基づき，43 種類の機能クラスに分類されている．そのため，BigCloneBench はソースコード分類手法の評価に用いることが可能である．BigCloneBench は開発者ら

が約 6 万個のメソッドを目視確認して作成した信頼性の高い大規模ベンチマークであるため，本研究ではこのベンチマークを利用する．

3.2 比較対象のソースコード分類手法

2.2 では，ソースコード分類手法に広く利用されている三つのニューラルネットワークについて説明した．本研究では，それぞれのニューラルネットワークに対して二つのソースコード表現（ソースコードのトークン列または AST）の各々を学習させ，計 6 種類のソースコード分類手法の精度を比較する．そのため，本節ではその 6 種類のソースコード分類手法について説明する．

ソースコード表現には，トークン列と AST の他に制御フローグラフやデータフローグラフなどがある．この二つのソースコード表現を生成するためにはソースコードのコンパイルが必要である．しかし，**3.1** で説明したように BigCloneBench はオープンソースソフトウェアからメソッド単位でソースコードを収集したベンチマークであるため，BigCloneBench に含まれている個々のソースコードをコンパイルすることが困難である．そのため，本研究ではコンパイルが必要なソースコード表現である制御フローグラフやデータフローグラフを利用しない．

本研究で利用した深層学習フレームワークは PyTorch1.5.0(注1)，各ニューラルネットワークで用いる活性化関数は ReLU，損失関数は CrossEntropy，最適化アルゴリズムは Adam である．各手法で用いるニューラルネットワークの隠れ層の層数とノード数は，グリッドサーチによって適切な値を決定した．グリッドサーチとは，ハイパーパラメータ群の候補を規則的に決め，各ハイパーパラメータの組合せを順番に探索し，適切なハイパーパラメータを決定する手法である．また，単語埋め込みベクトルは，次元数を増やすほどベクトルの質は良くなり次元数が 300 を超えるとベクトルの質の変化が少なくなる傾向がある [23] ため，本研究ではニューラルネットワークに入力する埋め込みベクトルの次元数を 300 に設定した．

3.2.1 FNN+Token

本手法では，**2.2.1** で説明した FNN に，Doc2Vec [24] により生成したトークン列のベクトルを学習させる．Doc2Vec は教師なし学習によって文書のベクトルを生成する手法である．ソースコードのトークン列はトークンの並び順に意味があるため，単語の前後関係を踏まえ，系列全体のベクトル化を行うことが可能な手法を利用する必要がある．よって本手法では Doc2Vec を利用してメソッドをベクトル化する．

具体的にはまず，学習データセット（**3.3** 参照）に含まれるメソッドを，javalang(注2) を用いてトークン列にする．次に，そのトークン列を文書，トークンを単語として扱い，Doc2Vec を用いてメソッドをベクトル化する．Doc2Vec ベクトルの次元数は 300，FNN の隠れ層は 4 層，隠れ層のノード数は 128 とした．

3.2.2 FNN+AST

本手法は，学習させるソースコード表現以外の設定は **3.2.1** で説明した FNN+Token と同じである．本手法では，**2.2.1** で説明した FNN に，Doc2Vec により生成した AST のベクトルを学習させる．具体的にはまず，学習データセット（**3.3** 参照）に含まれるメソッドを，Eclipse JDT(注3) の ASTParser を用いて AST に変換する．次に，AST ノードを深さ優先探索順に並べた列を文書，AST ノードを単語として扱い，Doc2Vec を用いてメソッドをベクトル化する．

3.2.3 LSTM+Token

本手法では，メソッドのトークン列を学習データとすることで，**2.2.2** で説明した LSTM にトークンの順序関係を学習させる．メソッドのトークン列は javalang を用いて生成する．LSTM の埋め込み層の次元数は 300，LSTM の隠れ層のノード数は 128 である．

3.2.4 LSTM+AST

本手法では，学習させるソースコード表現以外の設定は **3.2.3** で説明した LSTM+Token と同じである．本手法は，学習させるソースコード表現を AST ノードの深さ優先探索順列にすることで，LSTM に AST ノードの順序関係を学習させる．AST ノードの深さ優先探索順列は Eclipse JDT の ASTParser を用いて生成する．

3.2.5 GCN+Token

本手法では，**2.2.3** で説明した GCN にメソッドのトークン列を学習させる．GCN でトークン列を学習させるためにはトークン列をグラフで表現する必要があるため，今回はトークン列に含まれる各トークンをノードとして扱い，前後のトークンをエッジでつないだ 1 直線のグラフを扱う．GCN の実装には PyTorch

（注1）：https://pytorch.org/

（注2）：https://github.com/c2nes/javalang

（注3）：https://www.eclipse.org/jdt/

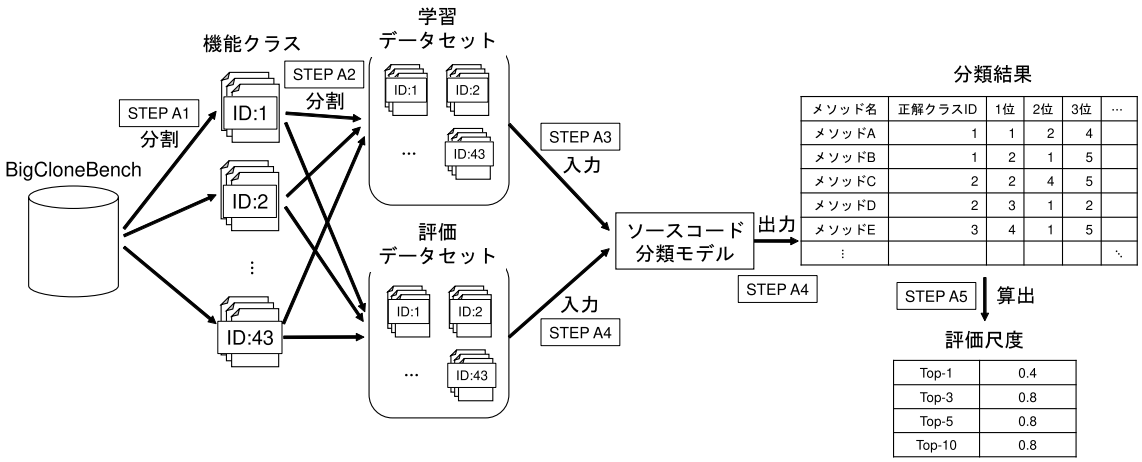


図 4　Top-k 算出方法の概要

Geometric(注4)を利用する．本手法におけるトークン列のグラフは無向グラフとみなし，エッジの重みは全て同じとする．ノードのベクトル化は Word2Vec [25] を用いて行う．具体的には，トークン列を文書，トークンを単語として扱い，Word2Vec を適用して各トークンのベクトルを生成し，各トークンに対応するノードに，生成したベクトルを割り当てる．メソッドのトークン列は javalang を用いて生成する．Word2Vec で生成するノードベクトルの次元数は 300，GCN の畳み込み層は 4 層，GCN の隠れ層のノード数は 128 とする．

3.2.6　GCN+AST

本手法では，学習させるソースコード表現以外の設定は **3.2.5** で説明した GCN+Token と同じである．本手法は，学習させるソースコード表現をメソッドの AST とする．これにより GCN は，ある AST ノードの周辺にどんなノードが出現するかを学習することができる．本手法における AST は無向グラフとみなし，エッジの重みは全て同じとする．AST ノードのベクトル化は Word2Vec [25] を用いて行う．具体的には，AST ノードを深さ優先探索順に並べ，各ノードを単語とみなし，Word2Vec を適用する．また，メソッドの AST は Eclipse JDT の ASTParser を用いて生成する．

ソースコードをグラフで表現する方法は AST の他に制御フローグラフやデータフローグラフがある．しかし，制御フローグラフやデータフローグラフを生成するにはソースコードのコンパイルが必要であり，**3.1** で説明した BigCloneBench にはコンパイルが困難なソースコードが含まれる．そのため本手法では，生成にソースコードのコンパイルが不要な AST を対象にする．

3.3　調 査 方 法

本研究では，ソースコード分類手法の評価尺度として Top-k を用いる．本研究における Top-k とは，各ソースコード分類手法が，評価データセットに含まれる各メソッドに対して機能クラスごとの分類確率を計算し，その確率が高いクラス順にランキングにしたとき，正解クラスが k 位以内に含まれている割合である．ここで正解クラスとは，各メソッドがもつ機能に基づいて，**3.1** で説明した BigCloneBench によって定められた機能クラスである．

Top-k の算出手順を図 4 に示す．Top-k の算出は以下の五つの手順に従って行う．

STEP A1
　3.1 で説明した BigCloneBench のメソッドを機能クラスごとに分割し，各機能クラスに固有の ID を割り当てる．

STEP A2
　各機能クラスのメソッドを 8:2 の割合でランダムに分割し，8 割を学習データセット，2 割を評価データセットとする．

STEP A3
　学習データセットのメソッドからソースコード表現を生成し，ニューラルネットワークに学習させる．

STEP A4
　学習データセットのメソッドを学習したニューラ

(注4)：https://github.com/rusty1s/pytorch_geometric

ルネットワークを用いて，評価データセットに含まれる各メソッドの分類を行う．

STEP A5

分類結果から Top-k を算出する．

本研究では，**3.2** で説明した 6 種類のソースコード分類手法に対して，以上の手順に従って Top-1，Top-3，Top-5，Top-10 をそれぞれ算出し比較する．ただし，手順の 2 番目では各機能クラスのメソッドの分割をランダムに行うため，分割の試行ごとに分類精度が変わる可能性がある．本研究では，6 種類の分類手法で同じデータセットを用いて学習と評価を行うために，共通のランダムシード値を用いて各機能クラスの分割を行う．これにより，高精度なソースコード分類を実現できるニューラルネットワークとソースコード表現の組合せについて調査する．

3.4 調査結果と考察

各ソースコード分類手法の精度を表 1 に示す．この表では Top-k で最も高い数値を太字で表している．表 1 から分かるように，各分類手法の中では，LSTM+Token が Top-1，Top-3，Top-5 で分類精度が最も高く，GCN+AST が Top-10 で分類精度が最も高いという結果になった．ソースコード分類では，Top-1 が優れていることが重要だと考えられる．**2.1** で挙げている機能タグの自動付与を実現するために，表 1 における LSTM+Token と同じ精度である「Top-1 が 0.943」の分類手法と，Top-10 が極端に高い「Top-10 が 1.0」の分類手法のどちらかを選択する必要があると仮定する．前者は，ソースコードに機能タグを 1 個だけ付与し，そのタグが 94.3% の確率で正しい手法で，後者は，ソースコードに機能タグを 10 個付与し，正しいタグがその中に必ず一つ存在する手法である．前者の「Top-1 が 0.943」である手法の方が間違ったタグを付与する可能性が非常に低いため，タグを自動で付与することによって既存ソースコードの再利用を容易にするという目的に適していると考えられる．以上の理由からソースコード分類では主に Top-1 の高い手法を採用するのが望ましいため，本研究の調査結果からは，Top-1 の精度が最も高い LSTM+Token が最も優れたソースコード分類手法であり，LSTM+AST や GCN+AST も比較的優れたソースコード分類手法であることが分かった．

表 1　各分類手法の分類精度

分類手法	深層学習	Top-1	Top-3	Top-5	Top-10
FNN+Token	○	0.575	0.766	0.830	0.911
FNN+AST	○	0.644	0.803	0.853	0.922
LSTM+Token	○	**0.943**	**0.980**	**0.985**	0.991
LSTM+AST	○	0.939	0.977	0.981	0.991
GCN+Token	○	0.772	0.927	0.967	0.989
GCN+AST	○	0.803	0.948	0.972	**0.993**
FaCoY	×	0.840	0.940	0.950	0.958
Siamese	×	0.848	0.897	0.908	0.925

また，分類精度は，学習させるソースコード表現よりも用いるニューラルネットワークに大きく影響されることが分かった．LSTM を用いた手法は，学習させるソースコード表現にかかわらず分類精度が平均的に高かった．GCN を用いた手法は，Top-1 が LSTM を用いた手法に大きく劣り，Top-3，Top-5 は LSTM を用いた手法と比べるとやや劣るが，Top-10 は LSTM を用いた手法と同じくらいの分類精度であり，FNN を用いた手法と比べると平均的に分類精度は高かった．FNN を用いた手法は，他のニューラルネットワークを用いた手法と比べると分類精度は低かった．よって，LSTM が最も高精度なソースコード分類を実現可能なニューラルネットワークであることが分かった．

次に，ニューラルネットワークとソースコード表現の組合せ方法について考察する．まず，FNN を用いた手法では，FNN+Token よりも FNN+AST の方が分類精度が高かった．このような結果になった原因は Doc2Vec にあると考えられる．Doc2Vec は *n*-gram の考え方をベースにした教師なし学習アルゴリズムで文書をベクトル化する．よって，トークン列の学習を行う際，括弧やセミコロンなど，ソースコードに頻繁に出現するトークンが Doc2Vec の学習のノイズになり，生成されるソースコードベクトルがソースコードの特徴をうまく表現できていなかったと考えられる．その一方で，AST の深さ優先探索順列には，学習のノイズになり得るノードなどの要素がトークン列より少ないため，トークン列に比べてソースコードの特徴を表現する良いソースコードベクトルを生成できたと考えられる．

次に，LSTM を用いた手法では，わずかに LSTM+Token の方が分類精度が高かった．LSTM+AST において，AST を LSTM に学習させるためには AST を何らかの系列に変換する必要があるため，今回は AST の深さ優先探索順列を学習させている．しかし，AST の深さ優先探索順列からソースコードを完全に復元することはできないため，元のソースコードの情報はわずかに欠落しており，その分だけ分類精度が低下したと考えられる．その一方で，LSTM+Token ではトークン列を学習するため，空白やインデントな

どのフォーマット情報を除き，ソースコードをそのまま学習可能である．また，LSTM は Doc2Vec と異なり，トークンの長期的な依存関係を学習することができる．そのため，LSTM の学習は，括弧やセミコロンなどの頻出トークンにあまり影響されないと考えられる．よって，AST よりもトークン列の方が LSTM との相性が良いと考えられる．

GCN を用いた手法では，GCN+Token よりも GCN+AST の方が分類精度が高かった．トークン列から生成したグラフには分岐がないため，グラフを学習可能なニューラルネットワークの利点を生かすことができていない．また，Doc2Vec の場合と同様に，頻出するトークンが学習ノイズになっている可能性が考えられる．その一方で，AST はグラフ形式の表現であるため，変形せずに GCN に学習させることができ，学習ノイズになり得るノードがトークン列に比べて少ない．よって，トークン列よりも AST の方が GCN との相性が良いと考えられる．

以上の結果から RQ の回答は，「LSTM とトークン列の組合せが最も高精度なソースコード分類を実現できる組合せであり，LSTM と AST の深さ優先探索順列の組合せや GCN と AST の組合せも比較的高い精度のソースコード分類を実現することができる」となった．

4. 深層学習を用いない分類手法との比較

3. では深層学習を用いる分類手法の精度比較を行い，LSTM を用いる手法が最も精度が高いことが分かった．しかし，深層学習を利用せずに高精度なソースコード分類ができる可能性がある．この前提を確認するために，本章では，**3.2** で説明した深層学習を用いるソースコード分類手法と，深層学習を用いないソースコード分類手法の中で代表的な手法の精度比較を行う．また，ソースコード分類に深層学習を用いる利点について考察する．

4.1 深層学習を用いない分類手法

本節では，深層学習を用いる手法との比較対象として選択した，深層学習を用いないソースコード分類手法について説明する．これらの手法は，構文的に類似したソースコードだけでなく意味的に類似したソースコードを検索できる最新手法であり，実装が公開されている(注5)(注6)ため，比較対象として選択した．

4.1.1 FaCoY

FaCoY [8] は，深層学習を用いない意味的類似ソースコード検索手法である．この手法は，開発者向け Q&A サイト StackOverflow(注7) を用いて類似したソースコードを検索する手法である．具体的に FaCoY はまず，入力として与えられたコード片に類似したソースコードが含まれる回答文を，ソースコードの類似度が高い順に n_s 個だけ StackOverflow から検索する．このとき，ソースコードの類似度は，二つのソースコードを TF-IDF（Term Frequency - Inverse Document Frequency）[26] でベクトル化したときの二つのベクトル間のコサイン類似度を用いて計算する．次に，検索された回答文に対応する質問文をクエリとし，再び StackOverflow から回答文の検索を行い，検索結果の先頭から n_q 個の回答文に含まれるソースコードを合計 n_c 個まで検索結果として出力する．このような手順で，FaCoY は入力として与えたコード片に類似するソースコードを検索する．この手法のキーアイデアは，“Q&A サイトにおいて類似した質問文に対応する形で出現するソースコードは意味的に類似している”であり，このアイデアによって構文的に類似したソースコードだけでなく意味的に類似したソースコードも検索することができる．この手法では，入力コード片と検索されたソースコードに対して TF-IDF を適用してベクトル化し，二つのベクトル間のコサイン類似度を算出し大きい順に並べることで，検索されたソースコードのランキングを作成する．

FaCoY のコード検索性能に影響するハイパーパラメータには，前段落で述べた n_s，n_q，n_c の三つがある．本研究では，$n_s = 3$，$n_q = 3$，$n_c = 100$ で比較を行う．この設定は，文献 [8] における FaCoY の BigCloneBench に対する適用実験で用いられたものである．

4.1.2 Siamese

Siamese [9] は，深層学習を用いない意味的類似ソースコード検索手法である．この手法は，n-gram を用いて類似したメソッドを検索する手法であり，BigCloneBench を用いた評価実験で，意味的に類似しているが構文的な類似度が低いメソッドのペアも高い精度で検索できることが確認されている．この手法ではまず，メソッドを 4 種類の異なる配列（トークン列，n-gram の配列，識別子・文字列・型を正規化した後の

（注5）：https://github.com/FalconLK/facoy
（注6）：https://github.com/UCL-CREST/Siamese
（注7）：https://stackoverflow.com/

n-gram の配列，括弧・セミコロン以外を正規化した後の *n*-gram の配列）で表現する．次に，検索対象メソッドと入力メソッドの組において，それぞれの表現方法に対してトークンや *n*-gram の出現頻度を考慮したスコア θ（0〜100%）を算出する．最後に，しきい値 T よりもスコア θ が高いメソッドの組を類似メソッドとして出力する．

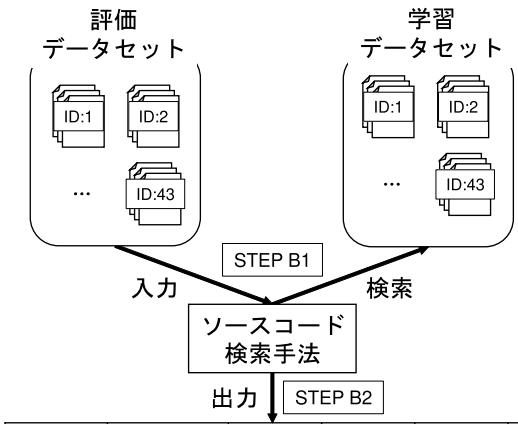
Siamese のコード検索性能に影響するハイパーパラメータには，*n*-gram を用いる際の n の値と前段落で述べたしきい値 T の二つがある．本研究では Siamese のデフォルト値である $n = 4$，$T = 10\%$ で設定し，比較を行う．

4.2 比較方法

4. では，深層学習を用いない意味的類似ソースコード検索手法をソースコード分類に適用し，**3.** で用いた 6 種類のソースコード分類手法と精度を比較する．**4.** で利用するベンチマーク及び評価尺度は，**3.** の比較調査と同様である．

本研究で選択した二つの意味的類似ソースコード検索手法において，検索クエリのソースコードと検索結果として出力されたソースコードは意味的に類似している．また，**2.1** で説明した本研究のソースコード分類の定義では，意味的に類似したソースコードは同じクラスに分類される．そのため本研究では，検索クエリのソースコードが，検索結果のソースコードが属するクラスに分類されるとみなすことで，意味的類似ソースコード検索手法をソースコード分類に適用する．また，検索結果のソースコードは，検索クエリとの類似度順にランキング化されて出力される．そこで本研究では，ランキングに含まれる検索結果のソースコードを，そのソースコードが属するクラスに置き換えることで，意味的類似ソースコード検索手法によって出力されたランキングをもとに，**3.3** で説明した Top-k を算出するためのランキングを作成する．

選択した意味的類似ソースコード検索手法をソースコード分類に適用し，**3.3** で説明した Top-k の算出に必要なランキングの作成手順を図 5 に示す．ランキング作成は以下の四つの手順に従って行う．ここで，以下の手順に出現する機能クラス ID は **3.3** の STEP A1 で BigCloneBench によって定められた機能クラスに対して割り当てた ID であり，学習データセット，評価データセットは **3.3** の STEP A2 で作成したデータセットである．また，図 5 の正解クラス ID は，評価データセットの各メソッドがもつ機能に基づいて，**3.1** で説明した BigCloneBench によって定められた機能クラスの ID である．



メソッド名	正解クラスID	1位	2位	3位	…
メソッドA	1	メソッドa	メソッドb	メソッドc	
メソッドB	1	メソッドa	メソッドc	メソッドb	
メソッドC	2	メソッドc	メソッドd	メソッドe	
メソッドD	2	メソッドc	メソッドd	メソッドb	
メソッドE	3	メソッドe	メソッドf	メソッドg	
⋮					⋱

機能クラスIDに置換 STEP B3

メソッド名	正解クラスID	1位	2位	3位	…
メソッドA	1	1	1	2	
メソッドB	1	1	2	3	
メソッドC	2	2	3	2	
メソッドD	2	2	3	1	
メソッドE	3	2	1	3	
⋮					⋱

重複削除 STEP B4

メソッド名	正解クラスID	1位	2位	3位	…
メソッドA	1	1	2	…	
メソッドB	1	1	2	3	
メソッドC	2	2	3	…	
メソッドD	2	2	3	1	
メソッドE	3	2	1	3	
⋮					⋱

図 5　意味的類似ソースコード検索手法を用いてランキングを作成する手順

STEP B1
　評価データセットに含まれる各メソッドを検索クエリに設定し，学習データセットに対して類似ソースコード検索を実行する．

STEP B2
　検索結果が出力される．このとき FaCoY の場合

は，検索結果のソースコードがランキング形式で出力される．Siamese の場合は，検索クエリの類似ソースコードがスコアとともに出力されるので，出力された類似ソースコードをスコアが高い順に並べることでランキングを作成する．

STEP B3

STEP B2 で得られたランキングに含まれる各ソースコードを，そのソースコードが属する機能クラスの ID に置き換える．

STEP B4

3.3 の STEP A4 で作成される分類結果では，同じ機能クラス ID は二つ以上重複して存在しない．したがって，Top-k の算出における条件を揃えるために，ランキングで重複した機能クラス ID のうち最も高い順位のもの以外を除外することで，再度ランキングを作成する．

以上の手順で作成したランキングを用いて Top-k を計算し，深層学習を用いる手法とソースコード分類精度を比較する．

4.3 比較結果と考察

深層学習を用いる分類手法と用いない分類手法の分類精度を表 1 に示す．この表の下 2 行は深層学習を用いないソースコード分類手法を示している．また，この表では Top-k で最も高い数値を太字で表している．

表 1 から分かるように，Top-1，Top-3，Top-5，Top-10 の全てで LSTM を用いる手法の精度は深層学習を用いない分類手法を上回った．LSTM を用いた手法のソースコード分類精度が深層学習を用いない手法の分類精度より高かった理由として，LSTM で学習可能な情報であるトークンや AST ノードの順序関係とその長期的な依存関係が BigCloneBench のソースコードを分類するのに重要だったことが考えられる．しかし，深層学習を用いない手法では，トークンやノードの関連性や長期的な依存関係を捉えた分類を行うことは困難である．Siamese は n-gram を用いているため，トークンの短期的な依存関係を捉えることはできるが，長期的な依存関係を捉えることはできない．そのため，分類精度が低くなったと考えられる．また，FaCoY はソースコードの類似性を捉えるために Q&A サイトにおける質問と回答を用いている．類似した回答文に含まれているソースコードに類似性があることは確かだが，この類似性が BigCloneBench のソースコードを分類するのに必要な類似性とは異なっていたために分類精度が低くなった可能性がある．

以上のように，LSTM を用いた分類手法は，本研究で選択した深層学習を用いない既存手法よりも高精度なソースコード分類を行うことができることが分かった．

5. 妥当性の脅威

本研究の妥当性の脅威として 4 点を挙げることができる．

一つ目は，比較対象に設定したニューラルネットワークの種類が少ない点である．本研究では広く利用されている三つのニューラルネットワークを調査対象としたが，ソースコード分類に適用可能なニューラルネットワークは他にも様々なものが考えられるため，今後検証していく必要がある．

二つ目は，ソースコード分類手法の精度比較調査を行う際に，一つのベンチマークを使用したため，調査結果が使用したベンチマークに強く依存している可能性がある．しかし，本研究で使用した BigCloneBench は約 6 万のメソッドを開発者らが手作業で確認することによって作成された非常に大規模なベンチマークであり，様々なソフトウェアのソースコードが含まれている．そのため，BigCloneBench を用いることでソースコード分類の一般的な評価を行うことができると考えられるが，今後は他のベンチマークを用いた比較調査を実施し，本研究と同様の傾向が得られるかを調査する必要がある．

三つ目は，今回の調査結果は様々な要因によって変わり得る点である．ニューラルネットワークの学習結果に影響する要因として，隠れ層の数やノード数などのハイパーパラメータ，活性化関数，損失関数，最適化アルゴリズム，データセットの分割方法が挙げられる．また，ソースコードをニューラルネットワークに入力するために実施する前処理の手法も，結果に影響する要因であると考えられる．FNN を使用した手法におけるメソッドのベクトル化手法や，GCN を使用した手法におけるノードのベクトル化手法が結果に影響すると考えられる．また，トークン列を学習する手法については，変数名を正規化することによって結果が変わると考えられる．AST を学習する手法については，使用する構文解析器によって出力される AST が異なるため，結果が変わると考えられる．本研究では，比較する分類手法の間で以上の要因を揃えているため，比較結果に一定の意味はあると考えられる．しかし，

あくまでも本研究の結果は **3.** 及び **4.** で説明した設定のもとでの結果であるため，今後は，本研究で調査しなかったパラメータ設定や前処理手法についても調査する必要がある．

四つ目は，サポートベクターマシン（SVM）やランダムフォレスト（RF）など，機械学習の手法との比較を行っていないため，機械学習の手法が LSTM より優れた分類精度になる可能性がある点である．この点に関しては，既存研究 [6], [7] にて既に深層学習を用いたソースコード分類手法と機械学習の手法のソースコード分類精度が比較されていたため，本研究で新たに比較を行うことはせず，機械学習を行わずに意味的類似ソースコードを検索可能な最新手法である FaCoY と Siamese との比較を行った．ただし，既存研究 [6], [7] で用いられたデータセットは BigCloneBench ではない．また，比較対象として選択されたのは SVM のみである．そのため，本研究の実験を行った場合，SVM や RF 等の機械学習の手法の方が優れた分類精度になる可能性があるため，今後調査する必要がある．

6. 関 連 研 究

近年，深層学習を用いてソースコードの解析を行う研究が発表されている．

まず我々は，過去に FNN を用いて類似コードブロックの検索を行う手法を提案した [15]．この研究では，FNN を用いたソースコード検索において，BoW と Doc2Vec の二つのソースコードベクトル化手法のどちらを用いた場合に精度が優れているかを比較しており，BoW が高い精度を示している．しかし，BigCloneBench はソースコード数が非常に多いデータセットであるため，本研究で BoW を採用した場合，BoW で生成したベクトルは次元数が膨大になり，学習に非常に時間がかかる．そのため，本研究では Doc2Vec を採用した．また，Saini ら [13] や Nafi ら [14] はソースコードメトリクスを用いてソースコードをベクトル化し，FNN を用いて類似ソースコード検出を行っている．Saini ら [13] は，深層学習を用いない既存手法と FNN を用いた提案手法の精度を比較し，提案手法の類似ソースコード検出精度が優れていることを示している．また，Nafi ら [14] は，生成したベクトルのコサイン類似度を計算する手法と FNN を用いた手法を比較し，FNN を用いた手法の方が類似ソースコード検出精度が優れていることを示している．本研究では，Saini ら [13] や Nafi ら [14] の研究のように深層学習を用いない既存手法との精度比較を行ったのに加え，ニューラルネットワークを利用した手法の精度比較を行った．

また，RNN を用いた研究が発表されている．Zhou ら [16] は深層学習を用いたソースコード補完手法を提案している．この手法では学習するソースコード表現としてクラス名・メソッド名・トークン列を利用し，トークン列の解析には RNN を用いている．White ら [17] は深層学習を用いた類似ソースコード検出手法を提案している．この手法ではトークンのベクトル化において RNN を利用し，このベクトルを用いて AST のノードをベクトル化することで，AST の類似性を判定している．これら二つの手法は様々なソースコード表現を学習することで精度向上を目的としているのに対し，本研究では LSTM で学習するソースコード表現を 1 種類に限定することで，ソースコード表現と分類精度の関連性調査を目的としている，

また，GCN 以外の方法で AST を学習に用いた研究が発表されている．Mou ら [6] は，AST を 2 分木に変形した後，AST ノードのベクトル表現を教師なし学習で作成し，畳み込みカーネルを AST 全体に対してスライドすることで構文木の特徴を学習する TBCNN という手法を提案し，この手法をソースコード分類に適用している．Mou ら [6] は，SVM，DNN（本研究における FNN），RNN と TBCNN の分類精度を比較し，TBCNN のソースコード分類精度が優れていることを示している．また，TBCNN を除き，最適なベクトル化手法を選択した場合は FNN，SVM，RNN の順に分類精度が高いという結果も示している．ここで比較対象に選択されている RNN は木構造に対して用いる再帰型オートエンコーダであり，LSTM とは異なる．それに対して本研究では，精度の高かった FNN に加え，Mou ら [6] が比較対象に選択していないニューラルネットワークである LSTM と GCN を比較対象として選択し，トークン列と AST ノードの深さ優先探索順列を学習させた．その結果，LSTM にトークン列を学習させる手法が最も分類精度が高いという結果になった．Zhang ら [7] は，ソースコードの AST をステートメントレベルに分割してそれぞれベクトル化してから Bi-directional Gated Recurrent Unit（Bi-GRU）[27] にステートメントベクトルのストリームを入力することでソースコードをベクトル化する ASTNN という手法を提案し，この手法をソースコード分類に適用している．Zhang ら [7] は，LSTM などのニューラルネットワークを用いた様々な手法や SVM と ASTNN のソー

スコード分類精度を比較している．ニューラルネットワークを単体で用いた手法や機械学習の手法については，トークン列に対する LSTM，SVM，制御フローグラフに対する GNN の順で分類精度が高いことを示している．GNN は GCN の前身となる，グラフを学習可能なニューラルネットワークである．また，彼ら [7] の研究では，LSTM に対するトークン列や，GNN に対する制御フローグラフやデータフローグラフなど，著者らによって適当だと考えられたニューラルネットワークとソースコード表現の組合せが比較対象に選択されている．それに対して本研究では，LSTM にグラフ由来のソースコード表現を学習させたり，GCN にトークン列由来のソースコード表現を学習させるなど，既存研究で比較対象に選択されていないニューラルネットワークとソースコード表現の組合せに対しても比較調査を行った．その結果，LSTM とトークン列の組合せなど，一般に既存研究で比較対象に選択されることの多い組合せは，選択されていない組合せに比べて分類精度が高いことが分かった．

また，GCN を用いた研究が発表されている．Hua ら [20] は，トークン列に LSTM を，AST に Tree-LSTM を，制御フローグラフに GCN を適用し，これら三つのニューラルネットワークを組み合わせることで，高い精度で類似ソースコード検出を行う FCCA という手法を提案した．そして評価実験では，深層学習を用いない既存手法や深層学習を用いた既存手法と比較して，FCCA の類似ソースコードの検出精度が優れていることを示している．また，FCCA は LSTM，Tree-LSTM，GCN の三つのニューラルネットワークを組み合わせて利用しているが，これらの三つのニューラルネットワークを個別に用いた場合の精度比較も行っている．この結果，LSTM，Tree-LSTM，GCN の順に類似ソースコードの検出精度が高いことが明らかになっている．Hua らの研究によって，トークン列と LSTM の組合せの類似ソースコード検出精度が高いことや，制御フローグラフと GCN の組合せの精度が比較的低いことが明らかになっているため，本研究では，彼らの研究の比較結果で最も高精度なトークン列と LSTM の組合せと，比較対象に選択されていない組合せを比較するため，トークン列，AST と FNN，LSTM，GCN を総当たりで組み合わせてソースコード分類精度の比較を行った．

以上のように関連研究では，**2.2** で説明した 3 種類のニューラルネットワークの比較や本研究で比較対象に選択したソースコード表現の比較は，Hua ら [20] によって LSTM を用いた手法の方が GCN を用いた手法よりも類似ソースコード検出精度が高いという結果が示されていたのみであった．そのため，本研究では **2.2** で説明した 3 種類のニューラルネットワークに 2 種類のソースコード表現を組み合わせた 6 種類のソースコード分類手法の精度を比較した．また，Saini ら [13] と Nafi ら [14] の研究では深層学習を用いない既存手法よりも FNN を用いた手法の方が類似ソースコード検出精度が高いという結果が示されていたが，本研究の調査結果によって，深層学習を用いない分類手法の方が FNN を用いた手法よりも精度が高い場合があることが分かった．

7. むすび

本研究では，高精度なソースコード分類を実現できるニューラルネットワークとソースコード表現の組合せについて調査するため，深層学習を用いた 6 種類の手法をソースコード分類に適用し，ベンチマークに BigCloneBench を，評価尺度に Top-k を用いてソースコード分類精度を比較した．その結果，LSTM にトークン列を学習させる手法が最も高精度なソースコード分類を実現でき，LSTM に AST の深さ優先探索順列を学習させる手法や GCN に AST を学習させる手法も，比較的高精度なソースコード分類を実現できた．それに対し，FNN を用いた手法の分類精度はあまり高くなかった．また，深層学習を用いる手法と深層学習を用いない手法のソースコード分類精度を比較した．その結果，LSTM を用いてソースコードの構造情報を学習する深層学習の手法は，本研究で選択した深層学習を用いない手法よりも高精度なソースコード分類を実現できることが分かった．

今後の課題として以下の点を挙げることができる．

- 今回用いたもの以外のニューラルネットワークやソースコード表現に対して分類精度の調査を行う必要がある．また，SVM や RF 等の機械学習の手法に対しても分類精度を調査する必要がある．
- ソースコード分類のベンチマークが BigCloneBench のみであるため，他のベンチマークを用いた比較を行い，同様の傾向が得られるか調査する必要がある．
- 本研究で調査しなかったパラメータ設定や前処理手法を採用した場合の分類精度を調査する必要がある．

謝辞 本研究は JSPS 科研費 JP18H04094, JP19K20240, JP20K11745 の助成を受けた.

文　献

[1] R. Hoffmann, J. Fogarty, and D.S. Weld, "Assieme: Finding and leveraging implicit references in a web search interface for programmers," Proc. UIST 2007, pp.13–22, Newport, Rhode Island, USA, Oct. 2007. DOI:10.1145/1294211.1294216

[2] K.T. Stolee, S. Elbaum, and D. Dobos, "Solving the search for source code," ACM Trans. Softw. Eng. Methodol, vol.23, no.3, pp.26:1–26:45, June 2014. DOI:10.1145/2581377

[3] G. Kavita and F. Romano, "C# or java? typescript or javascript? machine learning based classification of programming languages," GitHub, https://github.co/2Jif7Sg, accessed Nov. 2020.

[4] R. Yokomori, N. Yoshida, M. Noro, and K. Inoue, "Use-relationship based classification for software components," Proc. QuASoQ 2018, pp.59–66, Nara, Japan, Dec. 2018.

[5] S. Kawaguchi, P.K. Garg, M. Matsushita, and K. Inoue, "Mudablue: An automatic categorization system for open source repositories," J. Syst. Softw., vol.79, no.7, pp.939–953, 2006. DOI:10.1016/j.jss.2005.06.044

[6] L. Mou, G. Li, L. Zhang, T. Wang, and Z. Jin, "Convolutional neural networks over tree structures for programming language processing," Proc. AAAI 2016, pp.1287–1293, Phoenix, Arizona, USA, Feb. 2016. DOI:10.5555/3015812.3016002

[7] J. Zhang, X. Wang, H. Zhang, H. Sun, K. Wang, and X. Liu, "A novel neural source code representation based on abstrace syntax tree," Proc. ICSE 2019, pp.783–794, Montréal, QC, Canada, May 2019. DOI:10.1109/ICSE.2019.00086

[8] K. Kim, D. Kim, T.F. Bissyandé, E. Choi, L. Li, J. Klein, and Y.L. Traon, "FaCoY: A code-to-code search engine," Proc. ICSE 2018, pp.946–957, Gothenburg, Sweden, 2018. DOI:10.1145/3180155.3180187

[9] C. Ragkhitwetsagul and J. Krinke, "Siamese: scalable and incremental code clone search via multiple code representations," Empir. Softw. Eng. J., pp.2236–2284, Aug. 2019. DOI:10.1007/s10664-019-09697-7

[10] L. Deng and D. Yu, "Deep learning: Methods and applications," Found. Trends Signal Process., vol.7, no.3–4, pp.197–387, June 2014. DOI:10.1561/2000000039

[11] G. Cybenko, "Approximation by superpositions of a sigmoidal function," Math. Control Signals Systems, vol.2, pp.303–314, Dec. 1989. DOI:10.1007/BF02551274

[12] J. Schmidhuber, "Deep learning in neural networks: An overview," Neural Networks, vol.61, pp.85–117, 2015. DOI:10.1016/j.neunet.2014.09.003

[13] V. Saini, F. Farmahinifarahani, Y. Lu, P. Baldi, and C.V. Lopes, "Oreo: Detection of clones in the twilight zone," Proc. ESEC/FSE 2018, pp.354–365, Lake Buena Vista, FL, USA, Oct. 2018. DOI:10.1145/3236024.3236026

[14] K.W. Nafi, T.S. Kar, B. Roy, C.K. Roy, and K.A. Schneider, "Clcdsa: Cross language code clone detection using syntactical features and api documentation," Proc. ASE 2019, pp.1026–1037, San Diego, CA, USA, Nov. 2019. DOI:10.1109/ASE.2019.00099

[15] 藤原裕士, 崔　恩瀞, 吉田則裕, 井上克郎, "順伝播型ニューラルネットワークを用いた類似コードブロック検索の試み," ソフトウェアエンジニアリングシンポジウム 2018 論文集, pp.24–33, Tokyo, Japan, Aug. 2018.

[16] S. Zhou, H. Zhong, and B. Shen, "Slampa: Recommending code snippets with statistical language model," Proc. APSEC 2018, pp.79–88, Nara, Japan, Dec. 2018. DOI:10.1109/APSEC.2018.00022

[17] M. White, M. Tufano, C. Vendome, and D. Poshyvanyk, "Deep learning code fragments for code clone detection," Proc. ASE 2016, pp.87–98, Singapore, Singapore, Sept. 2016. DOI:10.1145/2970276.2970326

[18] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural Computation, vol.9, no.8, pp.1735–1780, 1997. DOI:10.1162/neco.1997.9.8.1735

[19] M. Schlichtkrull, T.N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," Proc. ESWC 2018, pp.593–607, Heraklion, Crete, Greece, June 2018. DOI:10.1007/978-3-319-93417-4_38

[20] W. Hua, Y. Sui, Y. Wan, G. Liu, and G. Xu, "Fcca: Hybrid code representation for functional clone detection using attention networks," IEEE Trans. Rel., pp.1–15, 2020. DOI:10.1109/TR.2020.3001918

[21] Y. Li, D. Tarlow, M. Brockschmidt, and R.S. Zemel, "Gated graph sequence neural networks," Proc. ICLR 2016 Poster Presentations, San Juan, Puerto Rico, May 2016. http://arxiv.org/abs/1511.05493

[22] J. Svajlenko, J.F. Islam, I. Keivanloo, C.K. Roy, and M.M. Mia, "Towards a big data curated benchmark of inter-project code clones," Proc. ICSME 2014, pp.476–480, Victoria, BC, Canada, Sept. 2014. DOI:10.1109/ICSME.2014.77

[23] O. Melamud, D. McClosky, S. Patwardhan, and M. Bansal, "The role of context types and dimensionality in learning word embeddings," Proc. NAACL 2016, pp.1030–1040, Association for Computational Linguistics, San Diego, California, June 2016. DOI:10.18653/v1/N16-1118

[24] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," Proc. ICML 2014, pp.II–1188–II–1196, Beijing, China, June 2014. DOI:10.5555/3044805.3045025

[25] T. Mikolov, I. Sutskever, K. Chen, G.S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," Proc. NIPS 2013, pp.3111–3119, Gardnerville, Douglas, Nevada, USA, Dec. 2013.

[26] G. Salton and M.J. McGill, Introduction to Modern Information Retrieval, McGraw-Hill, Inc., USA, 1986.

[27] D. Tang, B. Qin, and T. Liu, "Document modeling with gated recurrent neural network for sentiment classification," Proc. EMNLP 2015, pp.1422–1432, Lisbon, Portugal, Sept. 2015. DOI:10.18653/v1/D15-1167

（2020 年 11 月 7 日受付, 2021 年 3 月 25 日再受付, 4 月 5 日早期公開）

藤原　裕士

平成 29 年大阪大学基礎工学部情報科学科卒，令和元年度大阪大学大学院情報科学研究科博士前期課程了．現在は大阪大学大学院情報科学研究科博士後期課程に在学中．深層学習を用いたコードクローン分析手法に関する研究に従事．

崔　　恩瀞　（正員）

平成 27 年大阪大学大学院情報科学研究科博士後期課程了．同年同大学大学院国際公共政策研究科助教．平成 28 年奈良先端科学技術大学院大学情報科学研究科助教．平成 30 年より同大学先端科学技術研究科助教（改組による）．平成 30 年より京都工芸繊維大学情報工学・人間科学系助教．博士（情報科学）．コードクローン管理やリファクタリング支援手法に関する研究に従事．

吉田　則裕　（正員）

平成 21 年大阪大学大学院情報科学研究科博士後期課程了．同年日本学術振興会特別研究員（PD）．平成 22 年奈良先端科学技術大学院大学情報科学研究科助教．平成 26 年名古屋大学大学院情報科学研究科附属組込みシステム研究センター准教授．平成 29 年より同大学大学院情報学研究科附属組込みシステム研究センター准教授（改組による）．博士（情報科学）．コードクローン分析手法やリファクタリング支援手法に関する研究に従事．

井上　克郎　（正員：フェロー）

昭和 59 年大阪大学大学院基礎工学研究科博士後期課程了（工学博士）．同年大阪大学基礎工学部情報工学科助手．昭和 59 年～61 年，ハワイ大学マノア校コンピュータサイエンス学科助教授．平成 3 年大阪大学基礎工学部助教授．平成 7 年同学部教授．平成 14 年より大阪大学大学院情報科学研究科教授．ソフトウェア工学，特にコードクローンやコード検索などのプログラム分析や再利用技術の研究に従事．