Convolutional Neural Network を用いたフォールト数予測手法

小川 直記
株式会社 日立製作所
naoki.ogawa.od@hitachi.com

坂井 伸圭
株式会社 日立製作所
nobuyoshi.sakai.wx@hitachi.com

横内 弘
株式会社 日立製作所
hiroshi.yokouchi.dn@hitachi.com

要旨

ソフトウェア開発のテスト工程において，いかに品質を確保しつつ，テストの効率を上げるかは重要な課題の一つである．そのために，品質状況を判断し，品質と効率のバランスを最適化する必要がある．そこで，我々は品質状況を判断するための指標の一つとして，「摘出フォールト数の目標値」をテストの十分性を判断するために用いている．この際，高い精度で予測出来なかった場合は，過剰なテストを実施してしまう．しかし，高い精度の予測には開発者やテスト担当者が長年培ってきた経験が必要であるため，有識者に作業が集中してしまい全体のボトルネックとなってしまっていた．本研究では，過去のソフトウェア開発プロジェクトのデータから Convolutional Neural Network（以下 CNN と略す)を用いてフォールト数の予測を行い，定量的評価を行う．評価の結果，実際のプロジェクトで使用可能な精度でフォールト数の予測が可能なことがわかった．

1. はじめに

ソフトウェア開発において，テスト工程の工数が全工程に占める割合は大きく，例えば，新規開発の場合では結合テストと総合テストを合わせると，全工程の 28.6%を占め，改良開発においては 33.2%を占めるという報告[19]がある．このような中で，ソフトウェア品質を確保するためのテスト工程においては，実運用に耐えられる品質を確保しつつ，効率的にテストを実施することが必要である．

しかし，効率のためにテスト工数を減らすと，摘出フォールト数が少なくなるため，品質が下がり，品質確保に注力してテスト工数を増やすと，摘出フォールト数は頭打ちになるため，テストの効率が下がるというトレードオフの課題がある．

そこで，テスト工数を適切に割り当てるテスト戦略を立て，効率的に多くのフォールトを発見することが重要である[15]．

我々のユースケースベースのソフトウェア開発[9]でも，品質指標を用いてテスト戦略を立てており，テストの十分性を判断するための品質指標の一つとして，「摘出フォールト数の目標値」を用いてきた．「摘出フォールト数の目標値」は，テストの計画時に予測し，テスト後に実際にテストで摘出されたフォールト数と比較して，品質状況の判断に用いる．また，値の算出は，過去のプロジェクト情報や開発対象のユースケースごとの仕様書を基に，有識者の経験から推定する必要がある．そのため，有識者に作業が集中することでプロジェクトのボトルネックとなり，さらにフォールト数の予測値が有識者の経験に依存してしまう．

本研究では，ユースケースベースのソフトウェア開発を対象として，仕様書の情報と過去のプロジェクト情報から，ユースケースごとのフォールト数を予測する手法を提案する．過去のフォールト数と仕様書の文章データ，仕様書のページ数等の数値データを CNN[3]で学習させることで，フォールト数の予測を行う．本研究では，我々が実際に行った過去のソフトウェア開発プロジェクトのデータからフォールト数の予測を行い，定量的評価を行う．また，評価の結果，実際のプロジェクトで使用可能な程度の精度でフォールト数の予測が可能なことがわかった．

2. 関連研究

フォールト数の推定では，開発プロジェクトの工程ごとに，フォールト数を推定するモデルが提案されている．例えば，フォールト数の推移を時間や工数の関数として定義したレイリーモデルを使用する方法がある[12]．レイリーモデルでは開発工程ごとのフォールト数が予測できる利点はあるが，モデルの形状パラメータの設定が必要であり，モデルに即していない場合は利用できない．また，

形状パラメータを過去のプロジェクト実績から設定する方法も提案されている[16]が，ユースケースベース単位で設定する場合，過去のプロジェクト実績から適切なデータを抽出するのは難しい．ファイル単位でコードメトリクスや変更履歴から，フォールト数やフォールト密度を予測する手法[5]も存在するが，ファイルがどのユースケースで使用されるか推定することは難しいため，ユースケースベースでのフォールト数予測に適用することは難しい．他にも製品のフォールト数が推定できるシステムとして，SQE[1]が提案されている．しかし，ウォーターフォール型の画一的なソフトウェア開発時に製品全体への適用を想定されており，反復的かつユースケースベースの開発に適用するのは難しいことや，初期値の設定に経験が必要である．

また，品質を確保し，テストやレビューを効率的に行うために，フォールトを含んでいそうなモジュール(fault-prone module)を推定する研究は数多く行われている[2][18]．例えば，コードメトリクスや開発履歴のメトリクス，プログラムをテキストマイニングした結果を用いて，分類木[4]やロジスティック回帰，機械学習[6]で推定する手法が提案されている．また，fault-prone module の中でも Security[7][14]や Buffer Boundary Violation[17]等の特定の分野に特化した研究もおこなわれている．ただし，モジュール単位での推定であり，実際のプログラムが既に存在することが前提であることや，フォールト数の推定は行っていない．

3. CNN を用いたフォールト数予測手法

本研究では，有識者が仕様書の内容を読み，フォールト数の予測をすることを，CNN を用いて模倣する．

3.1. 従来のフォールト数予測

従来は，有識者がユースケースの規模と仕様書の内容から，経験をもとにフォールト数を予測していた．有識者がフォールト数を予測する場合は次のような作業を行う．まず，予め過去のユースケースの規模とフォールト数の実績値から，統計的に係数を算出しておく．予測する際に，算出された係数を用いて，新しいユースケースの規模に係数を掛け，予測値のベースとなる値を算出する．次に，有識者は仕様書の内容を読み，ユースケースの難しさを考慮し，経験によってベースとなる値を補正することでフォールト数を予測する．

3.2. CNN を用いた有識者の思考のモデル化

本研究では，ユースケースの規模と仕様書の内容から，経験をもとにフォールト数を予測していた有識者の思考を，CNN で模倣する．ユースケースの規模の情報として仕様書のページ数，ドキュメント数，単語数の数値データを使用し，ユースケースの難しさの情報して仕様書の文章データを使用する．ユースケースの規模の情報としては，他にも開発予定工数やコード量を使用することが考えられる．しかし，開発予定工数は有識者の経験で予測しているため，経験に依存する．また，コード量は異なるユースケースで使用されているコードがあることや，機能単位ではないため，コードとユースケースの対応付けが難しい．そのため，ユースケースの規模の情報として仕様書の数値データを用いる．有識者の思考を模倣するために，CNN を用いてユースケースごとの仕様書の数値データと仕様書の文章データの特徴を学習させる．ここで，本研究で入力として扱う仕様書とは，ユースケースごとに概要，外部仕様，内部仕様が記載されたものを指す．

3.3. CNN の構造

図 1 に本研究で用いた CNN の構造を示す．但し，各 Convolution Layer 及び Fully Connected Layer に対しての Dropout Layer は省略している．また，各レイヤの上にレイヤの名称と行，列，深さの順で次元数を記述する．但し，列，深さについて，存在しない場合は省略して記述する．本手法では活性化関数は全て ReLU 関数を用いている．Output Layer で ReLU 関数を用いた理由として，フォールト数は 0 件が最小値となるからである．また，最適化アルゴリズムには Adam[11]を使用し，コスト関数には最小二乗誤差を用いた．

3.4. 入力値の前処理

本手法では Input1，Input2 の入力値に対して，それぞれ次の前処理を行う．

- Input1

  入力する仕様書は，MeCab[20]を用いてわかち書きし，単語単位に区切る．仕様書の類似性を CNN に判別させるために，Word2Vec[8]を用いて 196 次元の単語の分散表現とする．また，仕様書を 500 単語ずつに分け，1 つの仕様書を複数の仕様書として扱う．例えば，2000 単語の仕様書があった場合，4 つの仕様書として扱う．また，500 単語に満たない場合，足りない部分は零ベクトルでパディングをする．CNN は主に画像処理で用いられる技術

であり，隣接情報を考慮した特徴を抽出することが出来る．自然言語処理では隣接した単語の特徴を抽出することで，単語の並び順を考慮した文脈の特徴を抽出が可能となることが期待できる．

- Input2

学習データに偏りが出ないように，仕様書のページ数，ドキュメント数，単語数の 3 つの数値データについて，平均 0，分散 1 となるように正規化を行う．Input2 は 1 つのユースケースに対して 1 つが定まるが，Input1 で 1 つの仕様書が複数の仕様書に分割される．この際には分割された仕様書に対して，Input1 は全て同じ値を入力とする．一般に CNN では多数の学習データが必要だが，多数のデータを得ることが難しい場合，疑似的にデータを増やすことが行われる．例えば，画像処理の分野では 1 枚の画像の明るさを変更したり，拡大，縮小，回転変形を加えたりし，データ数の問題を解決する．また，このようにデータを増やすことで，ノイズに強い学習モデルが生成できる．本研究では 1 つの仕様書を複数の仕様書に分割することで，同等の効果を得ることを目指す．

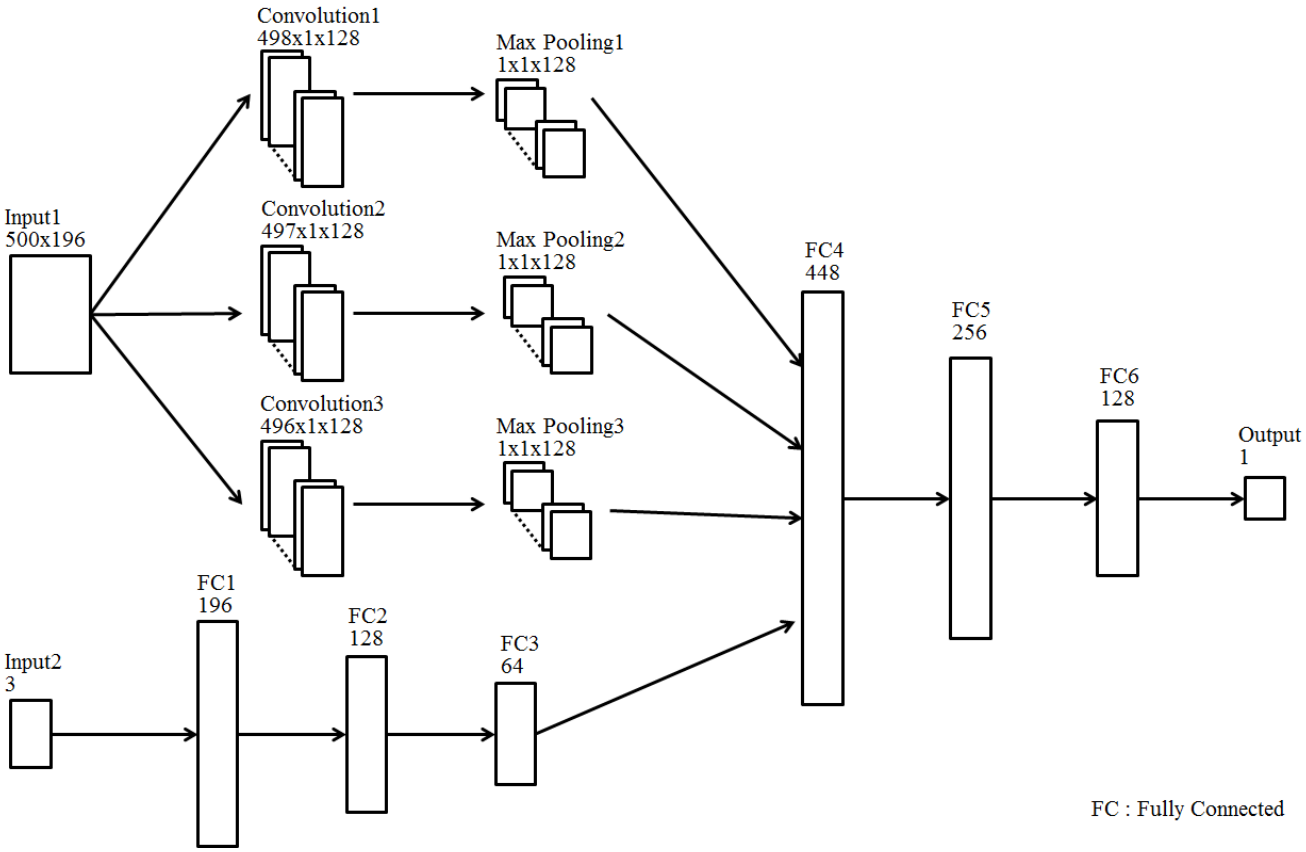

図 1：本手法における CNN の構造

4. 評価

4.1. 評価尺度

フォールト数予測精度の評価尺度として，MAE(Mean Absolute Error)，SD(Standard Deviation)の 2 つの指標を用いた．フォールト数の実測値をx，予測値を$\hat{x}$，データ数を N としたとき，それぞれの値は以下の式(1)，(2)で求められる．

$$\mathrm{MAE} = \frac{\sum |x - \hat{x}|}{N} \quad (1)$$

$$\mathrm{SD} = \sqrt{\frac{\sum (x - \hat{x})^2}{N}} \quad (2)$$

MAE は予測値のずれを表し, SD はユースケースごとにフォールト数の予測値がばらつかないかを示す指標であり，MAE，SD 共に小さい値は精度がよいことを表す.

4.2. 評価データ

本研究では, 開発プロセスや開発対象が類似している 2 つの製品 A, B を用いて検証を行った. 検証のために過去のプロジェクトのデータを学習データ, テストデータの 2 つに分けて精度の検証を行った. ここで, 学習データのバージョンはテストデータのバージョンより前のバージョンである. 学習データ, テストデータに用いたバージョン数とユースケース数をそれぞれ表 1, 表 2 に示す.

表 1 : 学習データのバージョン数とユースケース数

製品	バージョン数	ユースケース数
A	9	75
B	4	24

表 2 : テストデータのバージョン数とユースケース数

製品	バージョン数	ユースケース数
A	2	11
B	2	12

5. 評価結果

5.1. 結果

学習回数を 1500epoch とし, 1500epoch 目のモデルで製品 A, B のフォールト数を予測, 評価した結果を表 3 に示す. また, 仕様書の文章データと数値データを合わせた効果の検証のために, 仕様書の文章データのみから予測した結果と, 仕様書ページ数, ドキュメント数, 単語数の 3 つの数値データのみから予測した結果も併記し, それぞれ(仕), (数)と付記する.

表 3 : 同じプロジェクトの過去バージョンの学習データからフォールト数を予測した結果

学習製品	予測製品	MAE	SD
A	A	1.22	0.89
A(仕)		3.04	1.80
A(数)		7.35	11.58
B	B	3.72	2.16
B(仕)		4.09	2.51
B(数)		1.72	1.51

さらに, A と B を混ぜて学習させることで, 学習データ数を増やし, 予測精度が上げられないか検証した結果を表 4 に示す. ここで, A と B を混ぜたデータを A+B と表記する.

表 4 : 別プロジェクトのデータを混ぜた学習データからフォールト数を予測した結果

学習製品	予測製品	MAE	SD
A+B	A	1.27	0.85
A+B(仕)		3.61	2.29
A+B(数)		7.74	21.72
A+B	B	1.40	1.04
A+B(仕)		5.54	3.37
A+B(数)		2.37	4.48

次に, 学習の収束状況を評価するために, 製品 A, B のフォールト数予測における, 5epoch ごとの MAE, SD の値を図 2～5 に示す. ここで A+B(学)は本手法のモデルがフォールト数を予測するための表現力を有しているかを示すため, 学習データを用いてフォールト数を予測したものである.

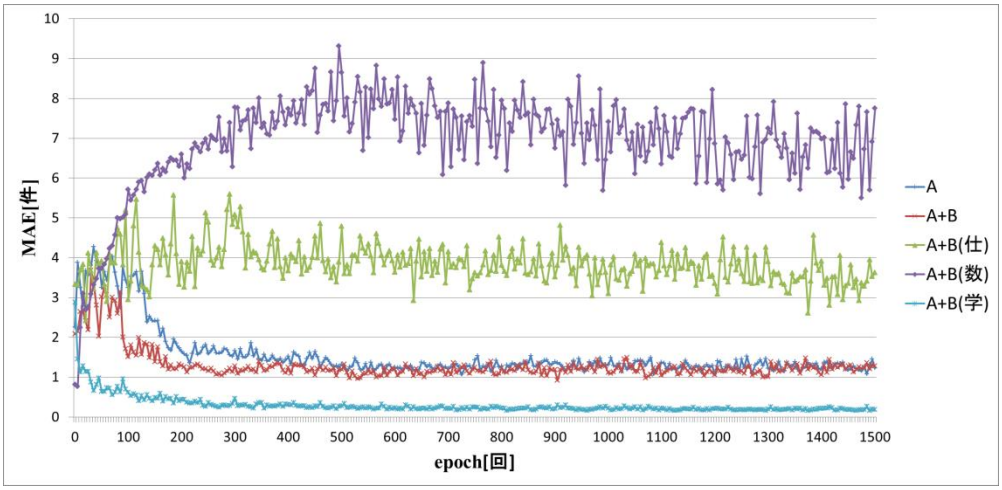

図 2：製品 A のフォールト数を予測した時の MAE の 5epoch ごとの推移

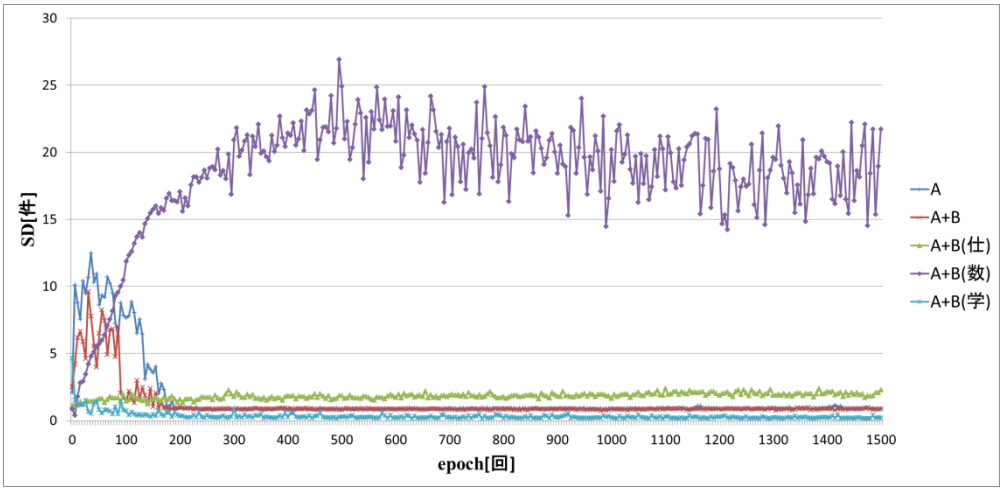

図 3：製品 A のフォールト数を予測した時の SD の 5epoch ごとの推移

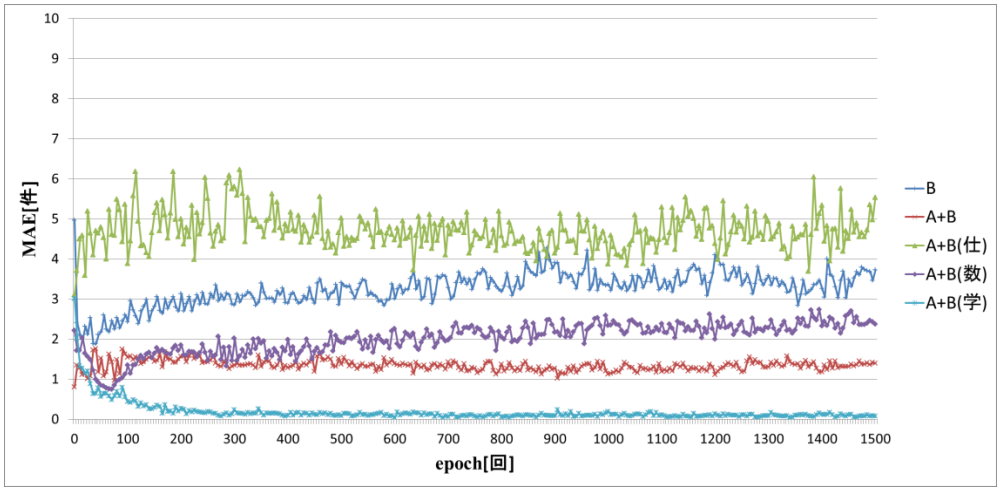

図 4：製品 B のフォールト数を予測した時の MAE の 5epoch ごとの推移

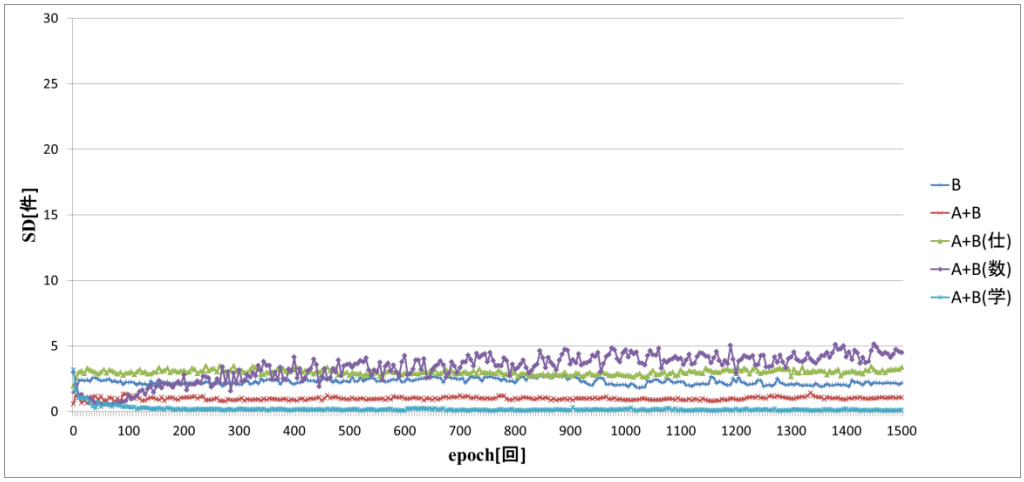

図 5：製品 B のフォールト数を予測した時の SD の 5epoch ごとの推移

6. 考察

6.1. 本手法の予測精度

表 3 より，製品 A では本手法の予測精度が最も高く，仕様書の文章データと数値データを合わせることで精度がよくなる効果が見て取れる．一方，製品 B では数値データのみが最も精度が高くなっている．製品 A，B では表 1 より，学習したユースケース数が 3 倍程度異なっているため，学習データ数が異なる．製品 B では学習データが十分でないため，学習しきれていない可能性がある．表 4 より，開発プロセス，開発対象が似ている A，B のデータを混ぜ，学習データ数を同じにして学習させた場合，A は予測値がほぼ同等であるが，B の予測値は改善が見られ，A，B 共に本手法の予測値が最も高くなっている．一方，仕様書のみや数値データのみの場合，A と B を混ぜると精度が悪くなっている．以上により，仕様書の文章データと数値データを合わせることで，どちらか一方だけでは表せないユースケースの特徴が存在し，本手法の CNN モデルで学習できていると考えられる．また，ユースケースの特徴は，開発プロセス，開発対象が似ている場合，共通している部分があると推測できる．そのため，製品 B は製品 A と比べ，学習ユースケース数が少なかったが，本手法のモデルを使用することで，少なかったデータを別プロジェクトのデータで補って，精度の高い予測ができたと考えられる．この結果は，異なるプロジェクト間で汎用的なモデルが生成できるということを示唆している．

ここで，参考までに，我々の有識者による予測値のデータから MAE と SD を算出した所，それぞれ 1.50[件]，1.98[件]であり，A と B を合わせて学習したモデルの場合の予測値では，有識者の精度以上となった．

6.2. 1500epoch での評価値の妥当性

本研究では，学習を 1500 回で打ち切って評価を行っている．しかし，固定回数での学習打ち切りでは，偶然精度の高い値になっただけの可能性がある．そこで，図 2，4 において MAE の値に注目すると，A+B を学習データとした時，200epoch 程度で小さな振動はあるが，一定値に収束していることがわかる．予測値の小さな振動は，過学習を抑えるために Dropout を実施していることが原因と考えられる．また，図 3，5 より SD の値に関しても，一定 epoch で収束している．よって，適当な回数で打ち切っても予測精度への影響は少なく，固定回数での打ち切りは実用上問題ないと考えられる．

6.3. モデルの妥当性

学習データに対して，MAE と SD が 0 に近づかなかったり，振動し続けていたりする場合，入力値に対して出力値を表現することが出来ないモデルであり，学習モデルとして妥当ではない．図 2～5 において，MAE，SD の値は 0 付近に収束しており，本研究における仕様書からフォールト数を予測するために妥当なモデルであるといえる．

6.4. 本手法の限界

本研究では，ある製品プロジェクトの過去のバージョンでフォールト数の予測を検証した．さらに，学習に使用していないプロジェクトのフォールト数を予測出来れば，適

用範囲が広がる. そこで, 製品 A, B をそれぞれ学習データとし, 製品 B, A のフォールト数をそれぞれ予測できるか検証を行った. しかし, 表 5に示す通り, 学習に使用していないプロジェクトへの適用は実現できなかった. 本研究では, 仕様書の文章データと, 仕様書のページ数, ドキュメント数, 単語数からフォールト数の予測を行ったが, コードメトリクス等の導入していないデータをいれることで, 学習に使用していないプロジェクトのフォールト数を予測できる可能性はあり, さらなる検証が必要である.

表 5: 学習に使用していないプロジェクトのフォールト数予測精度

学習製品	予測製品	MAE	SD
A	B	8.57	9.56
B	A	3.94	2.35

7. おわりに

本研究では, CNN を用いてフォールト数の予測モデルを構築し, 実際の開発プロジェクトのデータを用いて, ユースケースベースでのフォールト数予測を行い, 評価した. 評価の結果, 実際のプロジェクトで使用可能な精度でフォールト数の予測が可能ことを明らかにし, 有識者の経験に頼っていた部分を機械で置き換えられる可能性を示した.

また, CNN に学習させるデータが足りない場合, 開発プロセス, 開発対象が似ている別プロジェクトの製品のデータを混ぜて学習させることで, 精度を上げられることがわかった. 一方で, 学習に使用していないプロジェクトの予測へは適用できなかったため, さらなるモデルの改良や別の手法の適用を考える必要がある.

今後の課題として, 他プロジェクトや, ユースケースベースの開発以外でのプロジェクトでの検証を行うことが考えられる. 本研究ではユースケースベースの開発以外の仕様書を読み込ませ, フォールト数の予測ができるかは検証していないため, より汎用的に使える手法かの検証をしていく必要があると考えている.

また, 本手法では仕様書のみからのフォールト数の予測を行った. そこで, コードメトリクスや開発履歴[13]の使用していないデータを導入し検証することや, 仕様書等のドキュメントがない場合にでも適用できるフォールト数予測手法を考案していく必要があると考えている.

参考文献

[1] 橋本 弥一郎, 平島 俊一, 古賀 恵子, 横山 陽一, 森 文彦, ソフトウェア品質評価システム"SQE”, *日立評論*, vol. 68, No. 5(1986-5), pp.55-58 , 1986.

[2] Ohlsson, N., Alberg, H. Predicting fault-prone software modules in telephone switches. *IEEE Transactions on Software Engineering*. 886-894. 1996.

[3] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., Gradient-based learning applied to document recognition. *Proceedings of the IEEE*. 2278-2324. 1998.

[4] Khoshgoftaar, T. M., Allen, E. B., Deng, J. Using regression trees to classify fault-prone software modules. *IEEE Transactions on Reliability*. pp. 455-462. 2002

[5] Ostrand, Thomas J., Elaine J. Weyuker, Robert M. Bell. Predicting the location and number of faults in large software systems. *IEEE Transactions on Software Engineering*. pp. 340-355. 2005

[6] Kim, S., Whitehead Jr, E. J., Zhang, Y. Classifying software changes: Clean or buggy?. *IEEE Transactions on Software Engineering*. pp. 181-196. 2008.

[7] T. Zimmerman, N. Nagappan, L. Williams. Searching for a Needle in a Haystack: Predicting Security Vulnerabilities for Windows Vista. In *Proceedings of the 2010 Third International Conference on Software Testing , Verification and Validation (ICST '10).* pp. 421-428. 2010.

[8] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., Dean, J., Distributed Representations of Words and Phrases and their Compositionality. In *Advances in neural information processing systems*. pp. 3111-3119, 2013.

[9] Sakai, N., Nakanishi, M., Yokouchi, H., Improving User Experience in a Large-scale Software Development Project. In *Proceedings of 6th World Congress for Software Quality (6WCSQ)*. pp. 106-116. 2014.

[10] Kim, Y. Convolutional Neural Networks for Sentence Classification. *arXiv preprint arXiv:1408.5882*. 2014.

[11] Kingma, D., Ba, J., Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*. 2014.

[12] 安藝優子, 野中誠. 摘出済み欠陥を考慮したレイリーモデルに基づくソフトウェア欠陥予測手法. In *経*

営情報学会 全国研究発表大会要旨集 経営情報学会 2010 年秋季全国研究発表大会. pp. 27-27. 2010.

[13] 畑秀明, 水野修, 菊野亨. 不具合予測に関するメトリクスについての研究論文の系統的レビュー. *コンピュータ ソフトウェア*, pp.106-117. 2012

[14] Scandariato, R., Walden, J., Hovsepyan, A., Joosen, W. Predicting Vulnerable Software Components via Text Mining. *IEEE Transactions on Software Engineering*. pp. 993-1006. 2014.

[15] 中野大輔, 門田暁人, 亀井靖高, 松本健一. バグモジュール予測を用いたテスト工数割り当て戦略のシミュレーション. *コンピュータ ソフトウェア*. pp. 118-128. 2014.

[16] 田中公司, 生駒卓志, 津田和彦. ワイブル分布を用いたソフトウェア欠陥予測手法の検討. In *経営情報学会 全国研究発表大会要旨集 2015 年秋季全国研究発表大会* . pp. 21-24. 2015

[17] Meng, Q., Zhang, B., Feng, C., Tang, C. Detecting Buffer Boundary Violations Based on SVM. In *Information Science and Control Engineering (ICISCE), 2016 3rd International Conference on*. pp. 313-316. 2016.

[18] Kaur, I., Bajpai, N. An Empirical Study on Fault Prediction using Token-Based Approach. In *Proceedings of the International Conference on Advances in Information Communication Technology & Computing*. P.32 2016.

[19] ソフトウェア開発データ白書 2016-2107. 独立行政法人情報処理推進機構(IPA) 技術本部 ソフトウェア高信頼化センター(SEC). 2016

[20] MeCab: Yet Another Part-of-Speech and Morphological Analyzer, (http://taku910.github.io/mecab/)