Journal of the Korea Convergence Society
Vol. 12. No. 9, pp. 11-19, 2021

ISSN 2233-4890 / eISSN 2713-6353
https://doi.org/10.15207/JKCS.2021.12.9.011

CNN 모델을 이용한 프로그램 코드 변경 예측

김동관
목포해양대학교 해양컴퓨터공학과 교수

Predicting Program Code Changes Using a CNN Model

Dong Kwan Kim
Professor, Department of Computer Engineering, Mokpo National Maritime University

요 약 소프트웨어 시스템은 생명주기동안 기능 추가, 버그 수정, 새로운 컴퓨팅 환경 수용 등의 다양한 이유로 프로그램 코드 변경이 요구된다. 이러한 코드 수정 과정에서 새로운 오류 발생을 가져올 수 있으므로 프로그램 코드 수정 과정은 새로운 시스템 개발 못지 않게 신중하게 처리되야 한다. 또한, 오픈 소스 프로그램에 대한 재사용이 일반화된 소프트웨어 개발환경에서 오픈 소스 프로그램의 코드 변경 가능성을 예측할 수 있다면, 보다 양질의 프로그램 개발 효과를 기대할 수 있을 것이다. 본 논문은 소스 코드 변경을 예측하는 Convolutional Neural Network (CNN) 기반의 딥러닝 모델을 제안한다. 소스 코드 변경을 예측하는 문제는 딥러닝의 이진 분류 문제이며 레이블된 데이터가 요구되는 지도학습을 사용한다. 코드 예측 모델의 학습 및 시험을 위해 깃허브에서 수집한 Java 소스 코드와 코드 변경 로그를 데이터로 사용한다. 수집된 Java 소스 코드에서 소프트웨어 메트릭스를 계산한 후 제안된 코드 변경 예측 모델의 입력 데이터로 사용한다. 제안된 모델의 성능 평가를 위해 정밀도, 재현율, F1점수, 정확도가 측정되었으며 각각의 평가 지표에 있이서 CNN 모델은 95%, 다층 퍼셉트 기반의 DNN 모델은 92%를 달성했다.

주제어 : 소프트웨어 융합, 딥러닝, 코드 변경, 소프트웨어 메트릭스, 소프트웨어 유지보수

Abstract A software system is required to change during its life cycle due to various requirements such as adding functionalities, fixing bugs, and adjusting to new computing environments. Such program code modification should be considered as carefully as a new system development becase unexpected software errors could be introduced. In addition, when reusing open source programs, we can expect higher quality software if code changes of the open source program are predicted in advance. This paper proposes a Convolutional Neural Network (CNN)-based deep learning model to predict source code changes. In this paper, the prediction of code changes is considered as a kind of a binary classification problem in deep learning and labeled datasets are used for supervised learning. Java projects and code change logs are collected from GitHub for training and testing datasets. Software metrics are computed from the collected Java source code and they are used as input data for the proposed model to detect code changes. The performance of the proposed model has been measured by using evaluation metrics such as precision, recall, F1-score, and accuracy. The experimental results show the proposed CNN model has achieved 95% in terms of F1-Score and outperformed the multilayer percept-based DNN model whose F1-Score is 92%.

Key Words : software convergence, deep learning, code change proneness, software metrics, software maintenance

*This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2019R1I1A3A01056368).
*Corresponding Author : Dong Kwan Kim(dongkwan@mmu.ac.kr)
Received July 8, 2021 Revised July 30, 2021
Accepted September 20, 2021 Published September 28, 2021

1. 서론

지속적으로 변화하는 컴퓨팅 환경을 수용하기 위해 소프트웨어 시스템은 필수적으로 변화의 과정을 거치게 된다. 그러므로 소프트웨어 생명주기에서 소프트웨어 개발뿐만 아니라 유지보수 단계에 소요되는 비용과 노력을 간과할 수 없다. 소프트웨어 유지보수는 날로 복잡해지는 소프트웨어 시스템에 대한 변경 관리 및 제어 작업을 수행하며 실용적이지 못한 보수 작업은 기존 설계 작업의 부적절한 변경, 새로운 오류의 유입 등으로 인해 소프트웨어 품질 저하를 초래할 수 있다. 소프트웨어 시스템은 이해당사자들의 요구 사항에 대응하여 변화한다. 소프트웨어 활용 분야에 따라 차이가 있겠지만, 소프트웨어 시스템은 짧게는 수 개월에서, 길게는 수 십년 이상 코드 변경을 경험한다. 그러므로 소프트웨어공학 연구분야 중에 유지보수에 대한 내용이 큰 비중을 차지한다. 예를 들어, 소프트웨어 유지보수를 위한 기법, 도구, 알고리즘에 관한 연구들이 진행되고 있다. 소프트웨어 유지보수는 다양한 원인으로 발생된다. 소프트웨어에 포함된 오류의 원인으로 발생하기도 하며, 새로운 데이터나 운영체제, 하드웨어 환경의 변화에 적응하기 위해 코드 수정이 이루어지기도 한다. 또한, 기존 성능이나 유지보수성을 개선하기 위해 코드 변경이 필요한 경우도 있으며, 문제가 발생하기 전에 사전에 예방 차원에서 소프트웨어 코드 변경을 수행하기도 한다. 일반적으로 소프트웨어 코드 변경은 모든 코드에 일률적으로 발생하지 않으며, 일부 코드 부분에 발생하는 경향이 있다. 그러므로 소프트웨어 소스 코드 중 변경 가능성이 높은 부분을 사전에 알 수 있다면 유지보수 측면에서 보다 효과적이다. 유지보수 관심 대상을 코드 변경이 예측되는 부분으로 집중할 수 있기 때문이다. 예를 들어, 특정 클래스들에 대해 코드 변경이 예측된다면, 해당 클래스들의 유지보수를 위해 충분한 인력과 자원을 할당할 수 있을 것이다. 그러므로 소스 코드 변경 경향을 사전에 예측하기 위한 방법이나 실용적인 지침이 요구된다.

본 논문에서는 소스 코드 변경을 예측하기 위해 딥러닝 모델을 사용한다. 딥러닝 기술의 발전으로 인해 전통적인 소프트웨어공학 문제들을 해결하기 위해 심층 신경망을 이용한다. 코드 변경 예측 문제는 딥러닝의 전형적인 문제 유형인 이진 분류(Binary Classification)에 해당한다. 주어진 코드에 대해 코드변경이 발생할 것인지를 판단하는 문제이다. 본 논문에서는 지도학습 기반의 딥러닝 모델을 제안하며 이를 위해 레이블 정보를 포함한 데이터 세트를 구축한다. 깃허브와 같은 개방형 코드 저장소들을 통해 다양한 도메인의 프로그램 소스 코드를 수집할 수 있다. 수집된 소스 코드를 딥러닝 모델의 입력으로 사용하기 위해 코드 임베딩이나 특성 변수를 추출한다. 코드 변경 예측 모델은 구축된 데이터 세트를 통한 학습 과정에서 모델의 매개변수들을 스스로 결정한다. 본 논문은 이미지나 동영상 인식, 이미지 분류 등에서 의미있는 결과를 도출한 Convolutional Neural Network (CNN)을 이용한 코드 변경 예측 모델을 제안한다.

본 논문은 다음과 같이 구성된다. 2장에서는 관련 연구와 배경 기술을 소개한다. 3장에서는 소스 코드 변경 예측 시스템 개발을 위한 연구방법을 기술한다. 제안된 모델을 이용한 실험 결과와 이에 따른 분석은 4장에서 정리한다. 마지막으로 5장은 결론과 향후 연구 방향을 제시한다.

2. 관련 연구

코드 변경은 소프트웨어 시스템에서 피할 수 없는 작업단계로 코드 변경을 고려한 설계 기법들이 제안되고 있으며 향후 확장성을 고려한 시스템 개발을 권장하고 있다. 하지만, 코드 변경은 Barry Boehm [1]이 기술한 것처럼 소프트웨어 프로젝트 성공을 위해 관리되어야 하는 위험 요소이다. 즉, 코드 변경에 따른 노력과 비용이 크게 요구됨을 의미한다.

[2]는 소프트웨어 개발자들이 C++ 클래스의 변경 경향을 얼마나 효과적으로 예측하는가를 연구했으며, 자주 변경되는 클래스들의 특성화를 통해 보다 나은 예측 결과를 달성할 수 있음을 실험적으로 증명했다. [3]은 Java 소프트웨어 시스템의 변경 가능성을 예측하는 모델을 제시한다. 변경 가능성이 높은 파일을 예측하기 위해 다양한 객체 지향 메트릭스를 입력 데이터로 생성하고 기계학습 알고리즘을 사용한다. [4]는 오픈 소스 소프트웨어에서 변경 가능성이 높은 클래스를 식별하고 특성화하는 기법을 제안한다. 트리 기반 모델을 사용하여 변경 가능성이 높은 클래스를 식별하고 특성화한다. 이를 통해 개발자의 노력과 개발시간을 식별된 클래스에 집중하고, 특성화된 클래스별로 유사한 전략이나 기법을 적용한다. [5]는 객체 지향 시스템의 코드 변경 가능성을 예측하기 위한 실용적인 지침을 제공한다. 데이터 세트에서 종속 및 독립 변수를 추출하고 메트릭스 값을 추출한다. 데이터 세트는 훈련 및 시험데이터로 구분되며 분류 알고리

즘을 학습에 사용된다. 해당 연구는 클래스 변경 경향 예측을 위한 분류 모델을 개발하는 과정을 세부 단계별로 상세하게 기술하여 실용적인 지침을 제시한다. 또한, 불균형 데이터 세트를 처리하는 과정을 포함한다.

소스 코드 변경을 유발하는 원인은 다양하며 그 중 하나는 코드 스멜과의 연관성에 대한 연구이다 [6-8]. 코드 스멜은 프로그램 코드 중 가까운 장래에 심각한 문제를 발생시킬 수 있는 가능성을 가진 코드 블록을 의미하며 코드 스멜을 제거하는 것이 해당 프로그램의 유지보수 측면에서 효과적이다. 즉, 코드 스멜을 포함하는 코드 블록에 대한 코드 변경이 요구된다. [8]은 코드 스멜이 장래 코드 수정을 예측할 수 있는 중요한 지표임을 기계 학습 알고리즘을 적용하여 실험적으로 증명한다. 코드 스멜을 식별하고 리팩토링 작업을 통해 향후 유지보수에 효과적으로 기존 시스템을 변경한다.

코드 변경을 처리하기 위한 다양한 기법들이 제안되고 있으며 특히 코드 변경 탐지를 위해 머신러닝이나 딥러닝 알고리즘이 적용된다. 컴퓨터 비전, 이미지 처리, 데이터 마이닝, 언어 처리 등에서의 딥러닝 기술의 의미있는 결과물은 전통적인 소프트웨어공학의 문제에서도 새로운 돌파구를 제시하고 있다. 소프트웨어 생명주기 즉, 요구 사항 추출, 소프트웨어 설계, 구현, 소스 코드 모델링, 코드 자동 생성, 테스트 및 유지보수에 걸쳐 다양한 딥러닝 알고리즘이 탐색되고 적용된다[9-12]. 특히 딥러닝의 전형적인 응용 분야인 분류 문제에 대한 네트워크 모델 및 기법들이 소프트웨어 결함 및 버그 탐지, 보안 취약성 탐지, 코드 복제 탐지 등에서 실험적으로 향상된 결과를 보이고 있다.

소프트웨어공학에서는 소프트웨어를 보다 객관적으로 측정할 수 있는 방법에 대해 연구를 수행하고 있으며 그 중에 하나가 소프트웨어 메트릭스를 사용하는 것이다. 코드 메트릭스는 소프트웨어의 복잡도 등을 정량적으로 표현하는 방법으로 소프트웨어들을 비교하는 객관적인 특성이다. 소스 코드를 입력으로 사용하는 딥러닝 네트워크 모델에서 소스 코드의 특성은 메트릭스를 통해 추출될 수 있다. 예를 들어, 소프트웨어 결함 예측을 위해 다양한 딥러닝 모델이 연구되고 있으며 소스 코드는 메트릭스를 기반으로 표현된다 [7,13-15]. 또한, 소프트웨어 결함 예측과 함께 코드 스멜을 탐지하기 위해 코드 메트릭스를 사용한다 [6,16-18]. 소프트웨어 메트릭스는 Line of Code(LOC), McCabe[19], Halstead 메트릭스[20]와 같은 전형적인 메트릭스에서부터 객체지향 메트릭스 [21]를 포함한다. 또한, 프로그래밍 언어의 특성을 보다 적절하게 표현할 수 있는 메트릭스가 새롭게 정의되기도 한다.

본 논문에서는 코드 변경 예측을 분류 문제로 정의하고 딥러닝 네트워크 모델을 통해 주어진 코드에 대해 변경 여부를 예측한다. 또한, 클래스 코드의 변경 가능성을 예측하기 위해 메트릭스를 사용한다.

3. 연구 방법

3.1 코드 변경 예측 시스템

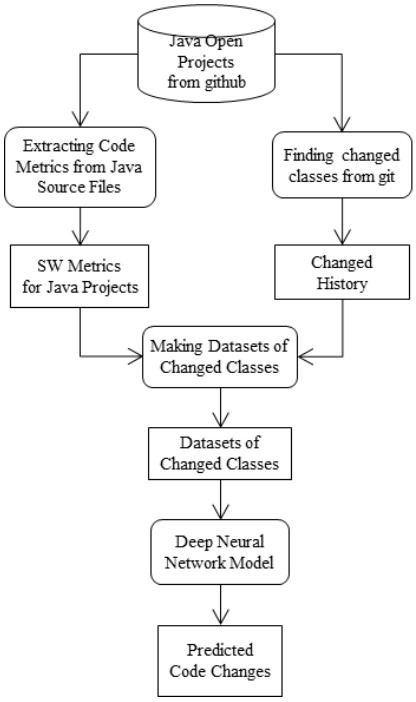

Fig. 1. The Overall Workflow of Predicting Code Changes Using Deep Learning.

Fig. 1은 Java 프로그램에서 변경이 예측되는 클래스를 찾기 위해 제안된 코드 변경 예측 시스템의 전체적인 작업흐름을 나타낸다. 코드 변경 예측 시스템은 깃허브 개방형 코드 저장소에서 다운로드한 Java 프로젝트를 사용한다. 깃허브에서는 프로젝트에 대한 코드 변경 히스토리가 관리되므로 해당 데이터를 이용하여 특정 클래스의 변경 히스토리를 알 수 있다. Java 소스 코드를 분석하여 코드 메트릭스를 계산한다. 사용된 메트릭스는 객체지향 메트릭스를 포함하며, 메트릭스 값을 계산하기

위해 RepositoryMiner 케이스 도구 [7]를 사용한다. 또한, 프로젝트의 코드 변경 히스토리를 분석하여 코드 변경을 경험한 클래스를 추출한다. 코드 변경 여부는 클래스들의 레이블 역할을 하며 코드 변경이 있었던 클래스는 '1'로, 그렇지 않은 클래스는 '0'으로 표현된다. 코드 메트릭스 계산과 레이블 추출 작업이 완료되면 딥러닝 모델의 입력으로 사용되는 데이터 세트가 구축된다. 해당 데이터로 학습한 코드 변경 예측 모델은 클래스의 메트릭스 값을 기준으로 Java 프로젝트의 클래스가 코드 변경한지 아닌지를 판단한다.

프로그램 코드 변경 가능성을 예측하기 위해 지도학습 기반의 딥러닝 네트워크 모델을 사용한다. 제안된 분류 모델의 지도학습을 위해 학습 및 시험 데이터 세트가 필요하며 표 1과 같은 프로그램 데이터가 사용된다. 깃 허브에서 다운로드한 10개의 Java 프로젝트의 소스 코드에서 분류 모델을 위한 특성을 추출한다. 다양한 도메인에 속하는 10개의 프로젝트를 대상으로 2016년 6월에서 2021년 5월까지 총 5년 동안 클래스 코드 변경 추이를 추적한다.

Table 1. Summary of the dataset. TC: Total of Classes, CC: Changed Classes, and NCC: Non-Changed Classes.

No.	Project	TC	CC	NCC	CCRatio(%)
1	ant	932	591	341	63.4
2	assertj	424	363	61	85.6
3	d14j	489	139	350	28.4
4	httpCore	369	143	226	38.8
5	mockito	324	207	117	63.9
6	springCore	375	240	135	64.0
7	springWeb	436	290	146	66.5
8	springContext	633	361	272	57.0
9	telephony	542	245	297	45.2
10	hibernate	3,032	505	2,527	16.7
Total		7,556	3,084	4,472	40.8

표 1은 각 프로젝트에 대한 전체 클래스 수, 코드 변경이 수행된 클래스 수, 코드 변경이 없는 클래스 수, 전체 클래스에 대한 코드 변경 클래스의 비율 등을 나타낸다. 전체적으로 40.8 %에 해당하는 클래스에 대해 소스 코드 변경이 발생했음을 알 수 있다.

Fig. 2는 데이터 세트에 포함된 Java 프로젝트의 클래스 변경 빈도를 나타낸다. 각 막대 그래프에서 x축은 코드 변경 빈도수를 구간으로 표시한다. 예를 들어, [1,5]는 1회에서 5회까지의 코드 변경 빈도수를 나타낸다. y축은 코드 변경이 발생한 클래스의 개수를 나타낸다. Ant 프로젝트의 경우를 보면, 코드 변경 빈도수 구간이 [1..5], [6,10], [11,5], [16,20], [20+] 총 5개 구간이다. [20+]는 코드 변경 빈도수가 20회를 초과하는 경우를 모두 포함한다. Ant 프로젝트에서는 [1,5] 구간에 592개의 클래스가 포함되며 [20+] 구간에는 21개 클래스가 포함된다. 대부분의 프로젝트가 [1,5] 구간에서 가장 많은 수의 클래스 변경이 발생하고 있음을 알 수 있다. assertJ 프로젝트의 경우는 [6,10] 구간에 속하는 클래스가 가장 많다. 딥러닝 모델의 학습 데이터 구축을 위해 True로 레이블되는 데이터는 일정 빈도수 이상(2회 또는 3회 이상) 변경된 클래스를 대상으로 하고 그 이하에 해당하는 클래스는 False로 레이블이 지정된다.

3.2 코드 메트릭스

Java 소스 코드를 분류 모델의 입력으로 사용하기 위해 다양한 표현 방법이 있으며 그 중에 하나가 코드 메트릭스이다. 소스 코드에서 사전에 정의된 코드 메트릭스 값을 계산한다. 계산된 코드 메트릭스는 csv 파일로 작성되며 코드 변경 예측 모델은 메트릭스 값을 사용하여 학습을 수행한다.

다음과 같은 코드 메트릭스가 주어진 클래스의 특성을 표현하기 위해 사용된다.

- ❑ LOC (Lines Of Code): 클래스에 속하는 모든 메소드의 코드 라인 수의 합계를 측정하는 메트릭
- ❑ NMD (Number of Methods Declared): 클래스에 선언된 메소드 개수를 측정하는 메트릭으로 일반적으로 객체 생성자와 단순한 getter/setter 메소드는 고려대상에서 제외시킴
- ❑ NAD (Number of Attributes Declared): 클래스에 선언된 속성(필드) 개수를 측정하는 메트릭
- ❑ LCOM5 (Lack of COhesion in Methods): 클래스 메소드들의 응집력을 측정하는 메트릭으로 속성을 접근하는 메소드의 수를 기준으로 함
- ❑ NADC (Number of Associated Data Classes): 연관 관계를 가진 데이터 클래스 수의 개수를 측정하는 메트릭
- ❑ ATFD (Access To Foreign Data): 특정 클래스에서 접근하는 외부 클래스의 속성의 수를 측정하는 메트릭
- ❑ WMC (Weighted Method Count): 클래스에 선언된 모든 메소드의 복잡도의 합을 측정하는 메트릭

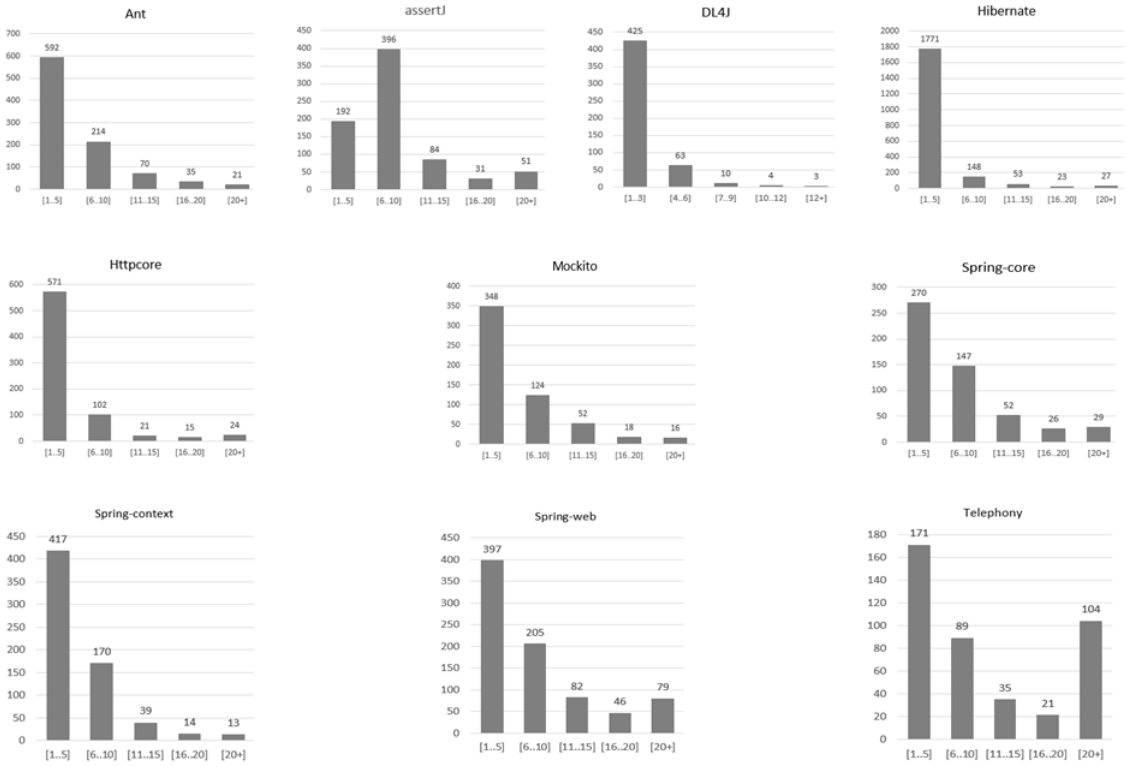

Fig. 2. Frequency Distribution of Code Chages of the Java Projects.

❑ 싸이클로매틱 복잡도(Cyclomatic Complexity): McCabe에 의해 고안된 프로그램 코드의 복잡도를 측정하는 메트릭으로 제어흐름그래프를 이용함. 싸이클로매틱 복잡도는 제어흐름그래프의 선형 독립 경로의 수 또는 평면 그래프의 영역의 수와 동일하므로 이를 이용하여 계산함.

상기에서 언급한 코드 메트릭스를 계산하기 위해 오픈소스 도구인 RepsitoryMiner를 일부 수정하여 사용하였다.

3.3 딥러닝 네트워크 모델

Fig. 3은 코드 변경 예측을 위해 제안된 딥러닝 신경망 모델의 구조를 표현한다. 코드 변경 예측 모델의 입력은 Java 파일들이므로 깃허브에서 다운로드된 파일 중에서 .java 파일들이 필터링된다. Java 파일은 메서드, 속

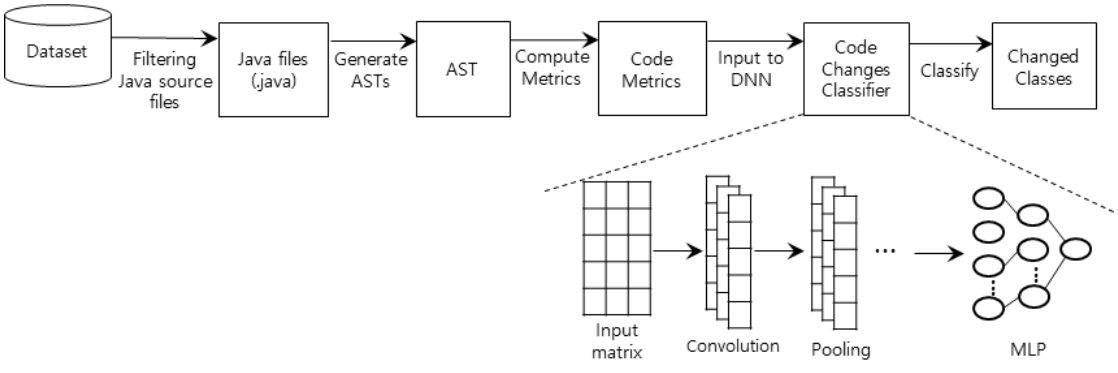

Fig. 3. The Proposed CNN-Based Code Change Prediction System.

성, 클래스 등으로 분석된다. 추출된 메소드와 클래스는 추상구문트리(Abstract Syntax Tree, AST) 형태로 변환한다. 소스 코드를 직접 분석하는 것보다는 추상구문트리 형태로 변환한 후 특성을 추출한다. 추상구문트리의 노드는 for문, if 문과 같은 소스 코드의 프로그래밍 언어 구성요소를 나타낸다.

제안된 코드 변경 예측 모델은 추상구문트리에서 사전에 정의된 코드 메트릭스를 계산한다. 코드 메트릭스는 클래스 레벨이나 메소드 레벨로 구성된다. Java 파일별로 메트릭스 값은 계산되며 실수형이나 정수형으로 표현된다. 추출된 메트릭스 값과 레이블로 구성된 데이터 세트의 레코드들은 코드 변경 분류기의 입력으로 사용된다. 딥러닝 모델의 학습 효과를 높이기 위해 입력 데이터에 대한 전처리를 수행한다. 예를 들어, 코드 라인수가 5줄 이하의 메소드는 데이터 세트에서 제외한다. 또한, 데이터 정규화 과정을 통해 모든 데이터 값이 [0..1]사이에 위치하도록 한다. 코드 변경 분류기는 CNN 딥러닝 모델을 기반으로 한다. 입력 계층에서 2차원의 메트릭스 값들을 받아들인다. 입력 값들은 코드 변경 분류기의 컨볼루션과 풀링 계층으로 구성된 히든 계층을 통과하고, 마지막 과정에서는 다층 퍼셉트론(MultiLayer Perceptron, MLP) 계층을 통과하게 된다. 히든 계층의 컨볼루션 계층에서 추출된 특징 값을 전형적인 인공 신경 지능망 구조인 다층 퍼셉트론 계층을 통해 분류 작업을 수행한다. 히든 계층은 여러 층의 컨볼루션과 풀링 계층을 포함할 수 있다. 제안된 모델에서는 1차원 컨볼루션 연산이 수행되며 활성화 함수로 ReLU를 사용한다. 컨볼루션 작업이 수행된 후 일련의 특성맵이 생성되며 이는 입력 데이터에 대한 특성들을 포함한다. 풀링 계층은 서브샘플링을 수행하는 계층으로 일정 영역 안에 속하는 값들 중 최대값이나 평균값을 취하여 입력 데이터의 차원(Dimentionality)를 줄인다. 결과적으로 계산량을 줄이는 효과를 기대할 수 있다. 또한, 데이터를 의도적으로 줄여서 손실을 발생시킴으로써 오버피팅을 방지하는 효과를 기대할 수 있다. 제안된 모델에서는 최대 풀링 알고리즘을 사용한다. 즉, 주어진 필터를 적용 후 가장 큰 값을 선택하고 나머지 작은 값들은 버린다. 이진 분류 모델이므로 출력 계층은 시그모이드(Sigmoid)를 활성화함수로 사용한다. 제안된 예측 모델의 오버피팅 문제에 대응하기 드롭아웃(Dropout) 계층을 사용한다. 드롭아웃은 뉴런을 임의로 삭제하면서 학습 과정을 수행함으로써 전체적으로 계산량을 줄이고 정규화를 효과적으로 달성하여 딥러닝 네트워크 모델의 일반화에 기여할 수 있다.

4. 실험 결과 및 분석

4.1 모델 평가 지표

제안된 코드 변경 예측을 위한 딥러닝 모델의 성능 평가를 위해 Table 2의 혼동 행렬을 사용한다. 혼동 행렬은 코드 변경 예측 분류 문제에서 예측한 값들과 실제 값들을 비교하여 4가지 경우를 고려한다. 혼동 행렬에서 TP는 실제 코드 변경이 발생한 클래스를 True로 예측한 경우를, FP는 코드 변경이 발생하지 않은 클래스를 코드 변경으로 잘 못 예측한 경우를 의미한다. 마찬가지로, FN은 실제 코드 변경이 발생한 클래스를 예측하지 못한 경우를, TN은 실제로 코드 변경이 발생하지 않은 클래스를 정확하게 예측한 경우를 나타낸다. 이러한 4개의 경우를 바탕으로 Table 3과 같이 딥러닝 모델의 대표적인 성능 평가 지표인 정밀도(Precison), 재현율(Recall), F1 점수(F1-Score), 정확도(Accuracy)를 계산한다.

Table 2. Confusion Matrix.

		Prediction Values	
		1	0
Actual Values	1	True Positive (TP)	False Negative (FN)
	0	False Positive (FP)	True Negative (TN)

코드 변경 예측 모델의 정밀도는 모델이 코드 변경이 발생한 것으로 예측한 클래스 중에서 실제로 코드 변경이 발생한 비율을 나타낸다. 재현율은 실제로 코드 변경이 발생한 클래스 중에서 예측 모델이 코드 변경이라고 예측한 클래스의 비율을 나타낸다. F1점수는 정밀도와 재현율의 조화평균으로 해당 값이 클수록 예측 모델의 성능이 좋다고 판단할 수 있다. 정확도는 시험용 입력 데이터를 얼마나 정확하게 예측했는지를 나타낸다. 혼동 행렬에서 대각선 즉, TP와 TN을 고려한다.

Table 3. Model Evaluation Metrics.

Eval. Metrics	Formula
Precision(P)	TP/(TP+FP)
Recall(R)	TP/(TP+FN)
F1-Score	2*(P*R)/(P+R)
Accuracy	(TP+TN)/(TP+FN+FP+TN)

4.2 실험 결과

본 논문의 실험을 위한 컴퓨팅 환경은 Table 4와 같다. GPU를 사용하고 있으며 대표적인 딥러닝 라이브러리인 케라스(Keras)를 사용한다. 실험에 사용된 전체 데이터 세트는 훈련용, 검증용, 시험용으로 분류되며 각각 3,477개, 870개, 1,087개로 총 5,434개 데이터가 사용되었다.

Table 4. Computing Environment for the Experiments.

Types	Items	Detail Info.
Hardware	CPU	Intel(R) Xeon(R) W-2123@3.6GHz
	GPU	NVIDIA GeForce RTX 2080
	RAM	32 GB
Software	OS	Microsoft Windows 10, 64 bits
	Python	Python 3.7.10
	Keras	2.3.1

Table 5는 실험에 사용된 CNN 모델의 딥러닝 계층을 나타낸다. 기본 블록으로 컨볼루션, 드랍아웃, 최대 풀링으로 구성된 3개의 계층이 반복해서 3개가 연결된 구조이다. 출력 계층은 완전 연결 계층인 dense 계층이 사용된다. 각 계층의 활성화 함수는 대표적인 ReLU를 사용하며 출력 계층은 이진 분류를 위해 sigmoid 함수를 사용한다.

Table 5. CNN Architecture Layers

Layer Type	Act. Function	#Filters
convolution dropout max_pooling	ReLU	32
convolution dropout max_pooling	ReLU	64
convolution dropout max_pooling	ReLU	128
dense	ReLU	-
dense	sigmoid	-

Fig. 4는 CNN 기반의 코드 변경 예측 모델에서 커널의 크기에 따른 검증용 정확도에 대한 추이를 시각화한다. 커널 크기가 6 이상 부터는 모델의 정확도가 큰 변화가 없지만 이전까지는 커널 크기가 모델의 정확도에 영향을 주고 있음을 알 수 있다. 커널은 CNN 모델에서 필터(Filter)라고도 하며 학습 대상이 커널 매개변수이다. 주어진 훈련 데이터에서 가중치에 해당하는 커널 매개변수를 학습한다. 커널은 지정된 간격으로 이동하면서 전체 입력데이터와 합성곱을 수행하여 특징맵을 생성한다.

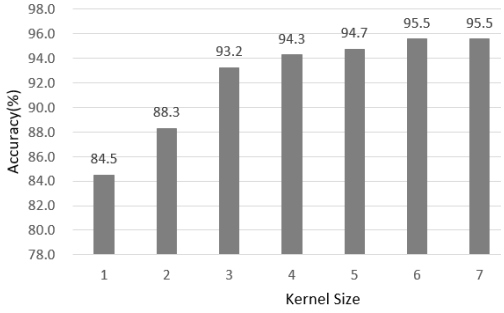

Fig. 4. Accuracy Distribution in terms of Kernel Sizes.

Fig. 5는 코드 변경 예측 모델의 훈련 과정의 손실과 정확도를 보여준다. 훈련 과정은 훈련용 데이터 세트를 이용한 단계와 검증용 데이터 세트를 이용하는 단계로 구성된다. 각 단계별로 손실 값과 정확도를 볼 수 있다. train_loss와 train_acc는 훈련 과정의 손실값과 정확도를 나타내고, val_loss와 val_acc는 검증 과정의 손실값과 정확도를 나타낸다. 에포크(Epoch)를 100으로 설정하였으며 에포크의 수가 100에 가까워짐에 따라 검증 과정의 손실 값은 0.1로, 정확도는 0.95에 수렴함을 알 수 있다. 전반적으로 모델 학습이 제대로 이루어지고 있다고 할 수 있다.

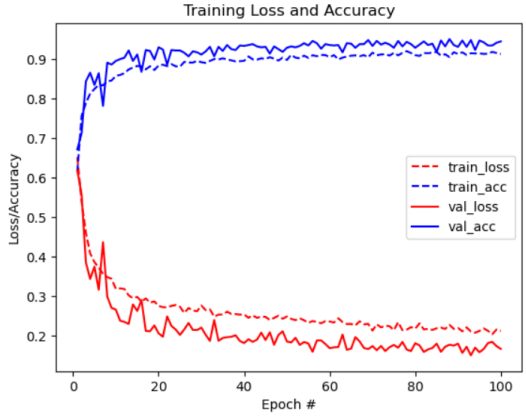

Fig. 5. Training Loss and Accuracy of the Code Change Prediciton Model

Table 6은 2개의 코드 변경 예측 모델의 성능을 앞에서 정의한 성과 지표에 따라 측정한 결과이다. DNN은 완전 연결 인공신경망(Fully-Connected DNN)을 나타내고 CNN은 CNN 기반 예측 모델을 나타낸다. 코드변

경 예측 모델의 입력 데이터 형상은 2차원으로 (5,434, 9)와 같다. 5,434개의 데이터가 각각 9개의 컬럼으로 구성되며 컬럼 9개는 8개의 메트릭스 값과 1개의 레이블으로 구성된다. 입력 데이터를 받아 들이는 모델의 최 앞단에 위치한 입력 계층에서는 입력 값의 형상을 지정한다. CNN 모델의 입력 계층에서는 입력 데이터 형상을 지정하는 매개변수인 input_shape를 “input_shape=(8, 1)”와 같이 지정한다. 즉, 메트릭스 개수에 따라 입력 형상이 결정된다. DNN 모델의 경우, 입력 계층인 케라스 라이브러리의 dense 계층의 매개변수 input_dim을 “input_dim=8”으로 지정한다.

F1점수 평가 지표를 볼 때, DNN 기반 예측 모델은 0.92이고 CNN 기반 예측 모델은 0.95임을 알 수 있다. CNN 기반 예측 모델이 일반적인 DNN 기반 예측 모델보다 성능이 우수하다. 평가 지표와 함께 모델 실행 시간을 측정한다. 실행 속도 측정을 위해 에포크: 100, 배치 크기: 4, 커널크기: 6로 설정하며 총 10번을 반복 실행한 후 평균 실행시간을 계산한다. 실험 결과에 따르면, DNN 모델은 에포크 당 3.5 초 소요하며 평균 실행시간은 346.67 초이다. CNN 모델에서는 에포크 당 6초 소요하며, 평균 실행시간은 607.46 초이다. CNN 모델이 DNN 모델에 비해 평균적으로 260.79 초 느리지만, 그 차이가 크지 않으므로 평가지표가 우세한 CNN 모델을 사용하는 것이 타당하다고 판단된다.

Table 6. Performance Measures of the fully-connected DNN and CNN. NCC: Non-Changed Code and CC: Changed Code.

	Precision		Recall		F1Score		Accuracy	
	DNN	CNN	DNN	CNN	DNN	CNN	DNN	CNN
NCC	0.92	0.93	0.90	0.96	0.91	0.94	0.92	0.95
CC	0.92	0.97	0.94	0.94	0.93	0.95		
Macro Avg.	0.92	0.95	0.92	0.95	0.92	0.95		
Weighted Avg.	0.92	0.95	0.92	0.95	0.92	0.95		

5. 결론 및 향후연구방향

본 논문은 소스 코드 변경을 예측하는 CNN기반의 딥러닝 모델을 제시한다. 모델의 학습 및 시험을 위해 Java 프로젝트의 코드 변경 히스토리를 기반으로 데이터 세트를 생성한다. 코드 변경이 발생한 클래스를 1로, 그렇지 않은 클래스를 0으로 설정한다. Java 소스코드의 특성을 표현하기 위해 코드 메트릭스를 계산한다. 모델의 성능 평가를 위해 정밀도, 재현율, F1점수, 정확도를 사용한다. CNN으로 구성된 코드 예측 모델의 경우 F1점수가 0.95이며, 다중 퍼셉트론으로 구성된 예측 모델의 경우는 0.92를 보였다. 또한, 에포크 크기와 커널 크기에 따라 CNN 기반 코드 변경 예측 모델의 성능에 영향이 있음을 알 수 있었다. 본 논문이 제공하는 코드 변경 예측 모델을 소프트웨어 유지보수나 오픈 소스 재사용 과정에서 활용할 수 있을 것으로 기대한다.

향후 연구 내용으로 코드 메트릭스 기반의 데이터 세트 생성 외에 소스 코드 임베딩을 이용할 수 있다. 소스 코드 임베딩을 이용하면 계산량이 증가할 수 있지만 보다 정교한 코드 변경 예측 모델을 개발할 수 있다. 또한, 보다 다양한 코드 메트릭스를 추가하여 코드 변경 예측 모델의 신뢰성을 보완할 수 있을 것이다.

REFERENCES

[1] B. W. Boehm. (1991). Software risk management: principles and practices. *IEEE Software 8(1)*, 32-41. DOI : 10.1109/52.62930

[2] M. Lindvall & K. Sandahl. (1998). How well do experienced software developers predict software change?, *Journal of Systems and Software, 43(1),* 19-27. DOI : 10.1016/S0164-1212(98)10019-5

[3] L. Kaur & A. Mishra. (2020). A Pragmatic Framework for Predicting Change Prone Files Using Machine Learning Techniques with Java-based Software, *Asia Pacific Journal of Information Systems, 30(3).* DOI : 10.14329/apjis.2020.30.3.457

[4] A. G. Koru & H. Liu. (2007). Identifying and characterizing change-prone classes in two large-scale opensource products. *Journal of Systems and Software*, *80(1),* 63-73. DOI : 10.1016/j.jss.2006.05.017

[5] C. S. Melo, M. M. L. da Cruz, A. D. F. Martins, T. Matos, J. M. da Silva Monteiro Filho & J. de Castro Machado, (2019). A practical guide to support change-proneness prediction. *In Proceedings of the 21st International Conference on Enterprise Information Systems*, SciTePress, 269-276. DOI : 10.5220/0007727702690276

[6] A. D. F. Martins, C. Melo, J. M. S. Monteiro & J. C. Machado, (2020). Empirical Study about Class Change Proneness Prediction using Software Metrics and Code Smells, *In Proceedings of the 22nd International Conference on Enterprise Information Systems*,

SciTePress, 140-147.
DOI : 10.5220/0009410601400147

[7] A. Barbez, F. Khomh & Y. Guéhéneuc. (2019). Deep Learning Anti-patterns from Code Metrics History. *In Proceedings of the 37th International Conference on Software Maintenance and Evolution*. IEEE, 114-124.
DOI : 10.1109/ICSME.2019.00021

[8] N. Pritam, M. Khari, L. Hoang Son, R. Kumar, S. Jha, I. Priyadarshini, M. Abdel-Basset, & H. Viet Long. (2019). Assessment of code smell for predicting class change proneness using machine learning. *IEEE Access*, *7*, 37414-37425.
DOI : 10.1109/ACCESS.2019.2905133

[9] F. Pudlitz, F. Brokhausen, & A. Vogelsang. (2019). Extraction of system states from natural language requirements. *In Proceedings of the 27th International Conference on Requirements Engineering. IEEE,* 211-222.
DOI : 10.14279/depositonce-8717

[10] J. Chen, C, Chen, Z. Xing, X. Xia, L. Zhu, J. Grundy, & J. Wang. (2020). Wireframe-based UI design search through image autoencoder. *ACM Transactions on Software Engineering and Methodology*, *29(3), ACM*, 1-31.
DOI : 10.1145/3391613

[11] Y. Hussain, Z. Huang, Y. Zhou, & S. Wang. (2020). CodeGRU: Context-aware deep learning with gated recurrent unit for source code modeling. *Information and Software Technology*, *125,* 106309.
DOI : 10.1016/j.infsof.2020.106309

[12] Y. Yang, X. Xia, D. Lo, & J. Grundy. (2020), A Survey on Deep Learning for Software Engineering, *ArXiv, 2011.*14597.

[13] Tianchi Zhou, Xiaobing Sun, Xin Xia, Bin Li & Xiang Chen. (2019). Improving defect prediction with deep forest. *Information and Software Technology*, *114,* 204-216.
DOI : 10.1016/j.infsof.2019.07.003

[14] H. K. Dam, T. Pham, S. W. Ng, T. Tran, J. Grundy, A. Ghose, T. Kim & C. Kim. (2019). Lessons learned from using a deep tree-based model for software defect prediction in practice. *In Proceedings of the 16th International Conference on Mining Software Repositories. IEEE,* 46-57.
DOI : 10.1109/MSR.2019.00017

[15] M. Wen, R. Wu & S. Cheung. (2018). How well do change sequences predict defects? sequence learning from software changes. *IEEE Transactions on Software Engineering*, *46*, 1155-1175.
DOI : 10.1109/TSE.2018.2876256

[16] M. Y. Mhawish & M. Gupta. (2019) Generating Code-Smell Prediction Rules Using Decision Tree Algorithm and Software Metrics, *International Journal of Computer Sciences and Engineering*, *7(5),* 41-48.
DOI : 10.26438/ijcse/v7i5.4148

[17] T. Guggulothu & S. A. Moiz. (2020). Code smell detection using multi-label classification approach. *Software Quality Journal*, *28,* 1063-1086.
DOI : 10.1007/s11219-020-09498-y

[18] M. Y. Mhawish & M. Gupta. (2020). Predicting Code Smells and Analysis of Predictions: Using Machine Learning Techniques and Software Metrics. *Journal of Computer Science and Technology, 35,* 1428-1445.
DOI : 10.1007/s11390-020-0323-7

[19] H. Watson, T. J. McCabe, & D. R. Wallace. (1996). *Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric*. Gaithersburg, MD, USA: NIST, 1-114.
DOI : 10.6028/NIST.SP.500-235

[20] V. Y. Shen, S. D. Conte, & H. E. Dunsmore. (1983). Software science revisited: A critical analysis of the theory and its empirical support, *IEEE Transactions on Software Engineering*, *9(2),* 155-165.
DOI : 10.1109/TSE.1983.236460

[21] S. Chidamber, & C. Kemerer. (1994). A metrics suite for object oriented design, *IEEE Transaction on Software Engineering*, *20(6)*, 476-493.
DOI : 10.1109/32.295895

김 동 관(Dong Kwan Kim) [정회원]

· 1993년 2월 : 숭실대학교 전산학과(공학사)
· 1998년 2월 : 숭실대학교 전산학과(공학석사)
· 2009년 7월 : Virginia Tech 컴퓨터과학과(공학박사)
· 2013년 3월 ~ 현재 : 목포해양대학교 해양컴퓨터공학과 교수
· 관심분야 : 딥러닝 알고리즘, 빅 데이터 분석, 소프트웨어공학, 런타임 시스템
· E-Mail : dongkwan@mmu.ac.kr