Proceedings of the International Conference on
Formal Methods and Foundations of Artificial Intelligence
Eszterházy Károly Catholic University
Eger, Hungary, June 5–7, 2025
pp. 140–147
DOI: 10.17048/fmfai.2025.140

Applying Tree-Based Convolutional Neural Networks to classify design patterns*

Gábor Kusper[ab], Erik Zoltán Hidi[c], Krisztián Kusper[c], Zijian Győző Yang[d], Szabolcs Márien[c]

[a]Eszterházy Károly Catholic University
kusper.gabor@uni-eszterhazy.hu

[b]University of Debrecen
kusper.gabor@inf.unideb.hu

[c]InnovITech
hidieric@gmail.com, {krisztian.kusper,szabolcs.marien}@innovitech.hu

[d]ELTE Research Center for Linguistics
yang.zijian.gyozo@nytud.elte.hu

Abstract. Automatic detection and classification of design patterns are an increasingly relevant task in modern software engineering, as it directly contributes to improving code quality, readability, and maintainability. In this paper, we propose the application of a modified Tree-Based Convolutional Neural Network (TBCNN) architecture for the recognition of GoF design patterns in Java source code. The approach leverages Abstract Syntax Trees (ASTs) as structural representations of programs, where nodes are encoded by a pre-trained embedding model that captures semantic similarities between language keywords. The resulting vectorized ASTs are processed by the TBCNN, enabling the model to learn both structural and semantic features characteristic of design patterns. For training and evaluation, we collected a dataset of Java implementations of design patterns from GitHub repositories, resulting in approximately 500–600 samples per pattern. Experimental results demonstrate high classification accuracy, with average precision, recall, and F1-scores exceeding 98% across eight design patterns. These findings confirm the viability of tree-based deep learning methods for pattern recog-

*This research was supported by the grant 2018-1.1.1-MKI-2018-00200 “Creating an automated quality assurance service with refactoring solutions”.

nition in source code. However, the model shows limitations when applied to real-world production code, likely due to the restricted representativeness of the training data, which consists mainly of educational implementations.

Keywords: design patterns, Tree-Based CNN, source code analysis

1. Introduction

Design patterns provide proven, reusable solutions to recurring design problems and are widely used to improve code quality, readability, and maintainability [3]. Despite their importance, *automatic* recognition of design patterns in source code remains challenging. Traditional approaches often rely on handcrafted rules or classical machine learning over engineered features, which are costly to maintain and tend to generalize poorly across projects and coding styles [9]. Recent advances in learning on program structure suggest that models operating directly on Abstract Syntax Trees (ASTs) can capture both syntactic and structural regularities of code [8]. Building on these insights, we revisit design pattern classification through the lens of tree-based deep learning.

Our study is motivated by the early work of Márien on the decision structures underlying design patterns [4–7], which highlights that patterns exhibit distinctive structural signatures. Tree-Based Convolutional Neural Networks (TBCNNs) have shown promise in learning from ASTs without handcrafted features [8], but their applicability to *pattern-level* classification in real-world codebases has not been systematically assessed. This leaves a gap between rule-heavy detectors and structure-aware neural models.

We address the following question:

> **RQ:** Can a modified Tree-Based Convolutional Neural Network effectively and robustly classify Gang-of-Four (GoF) design patterns in Java source code from AST representations?

We focus on eight GoF patterns that are both prevalent in practice and structurally discriminative: *Adapter*, *Bridge*, *Builder*, *Decorator*, *Null Object*, *State*, *Strategy*, and *Template Method*. We target method- and class-level manifestations as they appear in compilable Java files.

We propose a two-stage pipeline. First, we *pre-train* a keyword/token embedding model on Java code to obtain semantically informed vector representations for AST nodes. Second, we feed vectorized ASTs to a *modified* TBCNN, in which we adjust the coefficient matrix computation within the tree-based convolution to better reflect heterogeneous child-role contributions. This design aims to couple semantic token proximity with structural pattern signals.

To study feasibility at scale, we curate a pattern-labeled corpus from public GitHub repositories by querying files whose names and directory contexts indicate any of the eight target patterns. While this yields substantial coverage per pattern, many examples are educational implementations. We therefore explicitly evaluate generalization and discuss limitations stemming from dataset representativeness.

This paper makes the following contributions:

1. We formulate and evaluate a structure-aware deep learning approach for design pattern classification that operates directly on ASTs using a modified TBCNN.

2. We construct a pattern-focused Java dataset covering eight GoF patterns and report comprehensive per-pattern metrics.

3. We provide an empirical analysis highlighting strong performance on academic style implementations and outlining the generalization gap to production code, thereby charting directions for hybrid and data-centric improvements.

Our experiments indicate that the proposed model achieves high precision/recall on the curated benchmark, supporting the viability of tree-based deep learning for pattern recognition. At the same time, we observe reduced robustness on production code, underscoring the need for richer, more diverse training data and for combining learned structural features with lightweight, interpretable constraints. These observations motivate future work on dataset expansion, transfer learning, and hybrid rule+ML detectors.

Section 2 provides a concise overview of related work; Section 3 details the models, the modified tree convolution, and describes the dataset; Section 4 presents experiments and results; Section 5 concludes with limitations and future work.

2. Related work

The task of detecting and classifying design patterns has been studied from different perspectives, ranging from rule-based analysis to machine learning methods. Several surveys and empirical studies have highlighted both the importance of the problem and the challenges of building reliable detectors.

A systematic review by Yarahmadi and Hasheminejad [10] provides a comprehensive overview of design pattern detection approaches, categorizing them into rule-based, metrics-based, graph-matching, and machine learning families. Their study points out that most existing techniques rely heavily on handcrafted features or structural rules, which limit generalization to diverse coding styles and large-scale industrial systems.

Early work by Alhusain et al. [1] explored the feasibility of machine learning for design pattern recognition by using feature extraction combined with neural networks. Although pioneering, their approach was constrained by small datasets and relatively simple features, which restricted robustness. Later, Chaturvedi et al. [2] applied classical machine learning algorithms such as decision trees, support vector machines, and artificial neural networks to detect design patterns. They demonstrated that machine learning can achieve competitive accuracy, but also confirmed that feature engineering is critical and often domain-specific.

Zanoni et al. [11] investigated the integration of graph-based structural analysis with machine learning. Their MARPLE-DPD framework models design pattern instances as graphs and applies supervised learning for classification. This work represents an important step toward combining structural representations with learning techniques, yet it was mainly validated on relatively small systems.

Overall, the literature suggests that machine learning approaches to design pattern detection are promising, but existing studies are often limited by hand-crafted features, small datasets, and simplified implementations. This motivates structure-aware deep learning methods, such as Tree-Based Convolutional Neural Networks (TBCNNs), which can directly exploit Abstract Syntax Trees (ASTs) without manual feature engineering. Our work builds on these insights by applying and modifying a TBCNN architecture for pattern-level classification, aiming to better capture the structural and semantic signatures of GoF design patterns in Java source code.

3. Models

As it was mentioned earlier, we use two separate models for our problem. The architectures of these models are represented on Figure 1 and on Figure 2. The first model is used only for learning the programming language. The architecture is simple: embedding layer, fully connected layer and softmax layer.

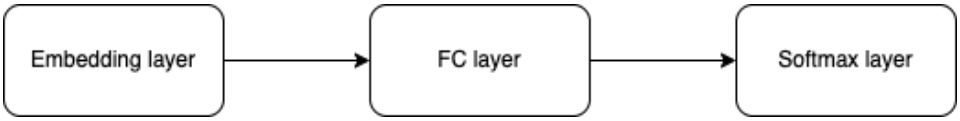

Figure 1. The keyword encoder.

The classification model has two input layers: vectorized AST and an array of node indexes. The inputs are concatenated, then the result is fed to the TBC layer, which is followed by a fully connected and a softmax layer.

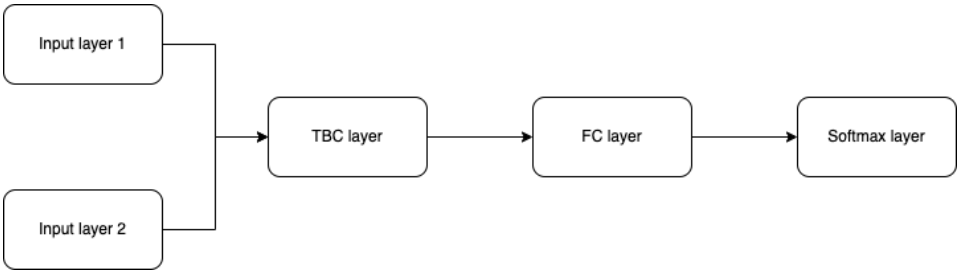

Figure 2. The design pattern classifier.

We modified the calculation of the coefficient matrix in the tree-based convolu-

tion. The equations (3.1), (3.2), and (3.3) represent the original calculations. We replaced equation (3.3) by equation (3.4).

$$\eta_i^t = \frac{d_i - 1}{d - 1} \tag{3.1}$$

$$\eta_i^r = (1 - \eta_i^t)\frac{p_i - 1}{n - 1} \tag{3.2}$$

$$\eta_i^l = (1 - \eta_i^t)(1 - \eta_i^r) \tag{3.3}$$

$$\eta_i^l = c_i \tag{3.4}$$

Where c_i is a vector with length equal to the number of children of the node filled with ones.

3.1. Training dataset

Source codes for the training dataset were gathered from GitHub, since the platform keep records of several million Java files. To select the required files, we used queries based on file names, thus downloaded Java files with names of design patterns. Approximately 500-600 files were gathered for each design pattern. The training dataset is built, but the quality may be questionable, since these files probably contain implementations for educational purposes, which makes them very similar.

4. Results

The dataset was divided into three parts: training, validation and test datasets. The proportion of files in the separate datasets is 70:15:15.

The model achieved high precision at the beginning already. After the 15th epoch, the precision is over 90% on both the training and validation datasets.

We evaluated the model successfully, the result is visible on Figures 3, 4, 5 and Table 1.

Table 1. Metrics.

DP	Precision	Recall	F1
Adapter	0.9894	1.0000	0.9947
Bridge	0.9500	1.0000	0.9744
Builder	1.0000	1.0000	1.0000
Decorator	0.9806	1.0000	0.9902
Null Object	0.9873	1.0000	0.9936
State	0.9753	0.9518	0.9634
Strategy	0.9800	0.9515	0.9655
Template Method	1.0000	0.9759	0.9878
Average	0.9828	0.9849	0.9837

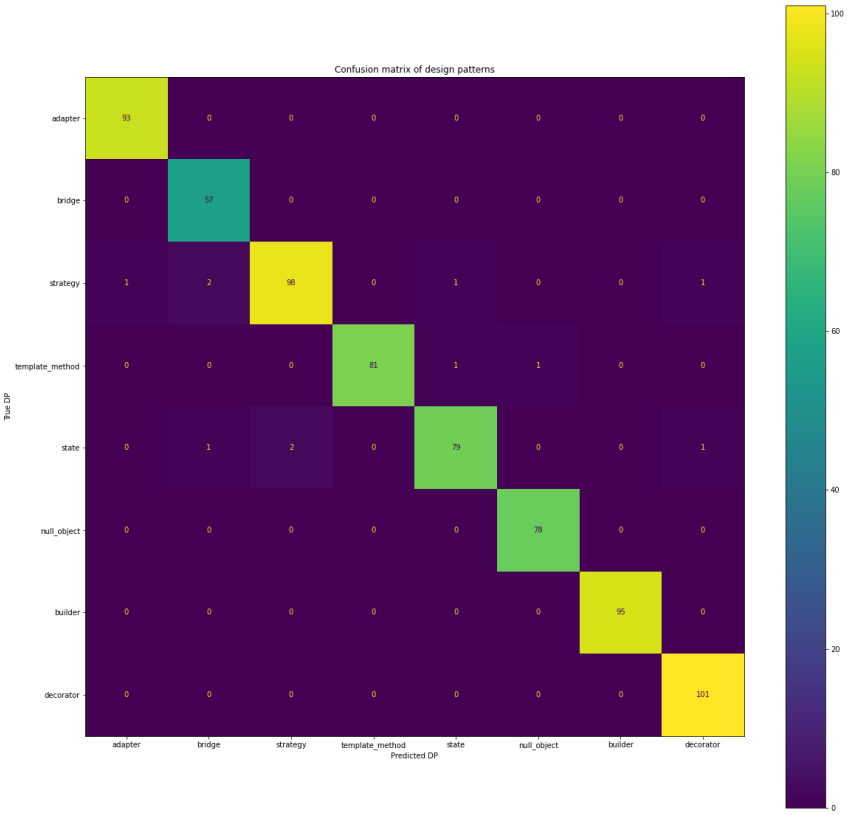

Figure 3. Confusion matrix of design pattern predictions.

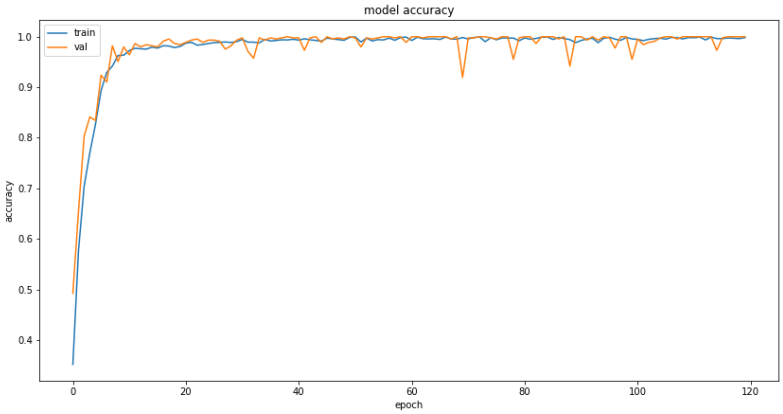

Figure 4. Training and validation accuracy across epochs.

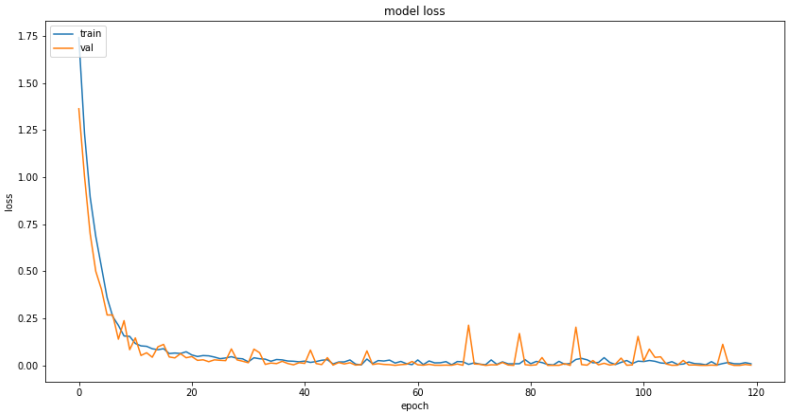

Figure 5. Training and validation loss across epochs.

5. Conclusion

In this paper, we applied a Tree-Based Convolution based machine learning model on a design pattern classification and detection problem with partial success. The model is very accurate in classifying the given design patterns, but as the experiments show it has problems with detecting design patterns in real world production code. We assume that the main problem behind this phenomenon is the lack of real world production source code in the training dataset.

Future work will focus on expanding the dataset with real-world industrial code bases and exploring hybrid approaches to improve generalization.

References

[1] S. Alhusain, S. Coupland, R. John, M. Kavanagh: *Towards machine learning based design pattern recognition*, in: 2013 13th UK Workshop on Computational Intelligence (UKCI), 2013, pp. 244–251.

[2] S. Chaturvedi, A. Chaturvedi, A. Tiwari, S. Agarwal: *Design pattern detection using machine learning techniques*, in: 2018 7th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO), 2018, pp. 1–6.

[3] E. Gamma, R. Helm, R. Johnson, J. Vlissides: *Design patterns: elements of reusable object-oriented software*, USA: Addison-Wesley Longman Publishing Co., Inc., 1995, isbn: 0201633612.

[4] S. Márien: *Decision based examination of object-oriented programming and Design Patterns*, Teaching Mathematics and Computer Science 6 (2008), pp. 83–109, doi: 10.5485/TMCS.2008.0174.

[5] S. Márien: *Decision based examination of object-oritented methodology using JML*, in: Annales Mathematicae et Informaticae, vol. 35, Eszterházy Károly College, Institute of Mathematics and Computer Science Eger, 2008, pp. 95–121.

[6] S. Márien: *Decision structure based object-oriented design principles*, in: Annales Mathematicae et Informaticae, vol. 47, 2017, pp. 149–176.

[7] S. Márien, G. Kusper: *Understanding Design Patterns as Constructive Proofs*, Proceedings of ICAI-2004 (2004), pp. 173–182.

[8] L. Mou, G. Li, L. Zhang, T. Wang, Z. Jin: *Convolutional neural networks over tree structures for programming language processing*, in: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, AAAI'16, Phoenix, Arizona: AAAI Press, 2016, pp. 1287–1293.

[9] H. Thaller, L. Linsbauer, A. Egyed: *Feature Maps: A Comprehensible Software Representation for Design Pattern Detection*, in: 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER), 2019, pp. 207–217.

[10] H. Yarahmadi, S. M. H. Hasheminejad: *Design pattern detection approaches: a systematic review of the literature*, Artificial Intelligence Review 53.8 (2020), pp. 5789–5846.

[11] M. Zanoni, F. A. Fontana, F. Stella: *On applying machine learning techniques for design pattern detection*, Journal of Systems and Software 103 (2015), pp. 102–117.