






# Transfer Learning for Just-in-Time Design Smells Prediction using Temporal Convolutional Networks

Pasquale Ardimento<sup>1</sup><sup>a</sup>, Lerina Aversano<sup>2</sup><sup>b</sup>, Mario Luca Bernardi<sup>2</sup><sup>c</sup>, Marta Cimitile<sup>3</sup><sup>d</sup> and Martina Iammarino<sup>2</sup><sup>e</sup>

<sup>1</sup>Computer Science Department, University of Bari Aldo Moro, Via E. Orabona 4, Bari, Italy

<sup>2</sup>University of Sannio, Benevento, Italy

<sup>3</sup>Unitelma Sapienza, University of Rome, Italy

**Keywords:** Design Smells Prediction, Software Quality, Deep Learning, Transfer Learning.


**Abstract:** This paper investigates whether the adoption of a transfer learning approach can be effective for just-in-time design smells prediction. The approach uses a variant of Temporal Convolutional Networks to predict design smells and a carefully selected fine-grained process and product metrics. The validation is performed on a dataset composed of three open-source systems and includes a comparison between transfer and direct learning. The hypothesis, which we want to verify, is that the proposed transfer learning approach is feasible to transfer the knowledge gained on mature systems to the system of interest to make reliable predictions even at the beginning of development when the available historical data is limited. The obtained results show that, when the class imbalance is high, the transfer learning provides F1-scores very close to the ones obtained by direct learning.


## 1 INTRODUCTION


In software engineering, design smells are defined as aspects that violate the fundamental principles of software design and negatively affect it (Suryanarayana et al., 2015). Unmanaged design smells can often lead to significant technical problems and increase the effort required for maintenance and evolution (Aversano. et al., 2020). Several approaches and tools are recently proposed to identify design smells (Alkharabsheh et al., 2018): some of these (Sharma et al., 2019; Al-Shaaby, 2020) use machine learning and deep learning techniques to perform both design smell detection and prediction.


Deep neural network approaches have proven to be effective in metric-based prediction tasks but are thwarted by software systems specific properties. First of all, software projects at the beginning of their development cycle, have insufficient data (Kitchen-


ham et al., 2007) to allow training of deep neural networks and even mature systems usually have imbalanced smelly revisions ratios making training steps very difficult or even unfeasible. This paper investigates whether the adoption of a transfer learning approach can be effective in mitigating both problems. Transfer learning (Lumini and Nanni, 2019) allows training a prediction model using knowledge gathered from other projects with known smells. This helps to obtain satisfying predictions even in the early stage of the development life cycle. However, to increase the data availability, system evolution data is collected at the commit level through the assessment of internal quality metrics and process development metrics. The specific process metrics allow us to also capture the software development process and the developers' behavior. This study proposes an adequate neural network architecture effective for the given prediction task. The reference architecture is a Temporal Convolutional Network (TCN): this kind of neural network is characterized by casualness in the convolution architecture design (Bai et al., 2018) making it suitable to our prediction problem where the causal relationships among the metrics evolution and design smells presence should be learned. To validate the

<sup>a</sup> <https://orcid.org/0000-0001-6134-2993>

<sup>b</sup> <https://orcid.org/0000-0003-2436-6835>

<sup>c</sup> <https://orcid.org/0000-0002-3223-7032>

<sup>d</sup> <https://orcid.org/0000-0003-2403-8313>

<sup>e</sup> <https://orcid.org/0000-0001-8025-733X>

proposed approach, we conducted an experiment on three well-known open-source software systems and we compare transfer and direct learning results.

In Section 2 some backgrounds about transfer learning and design smell are discussed. In Section 3 the related work is discussed. The proposed transfer/direct learning approach, the adopted features, the data extraction process and the TCN architecture are described in Section 4. The experiment description is provided in Section 5, while the experiment results are discussed in Section 6. Finally, in Section 7 and 8 respectively, the threats to validity and the conclusions are discussed.

## 2 BACKGROUND

### 2.1 Transfer Learning

In some real-world machine learning scenarios, training data and testing data are not taken from the same domain because there are cases where training data is expensive or difficult to collect. Therefore, there is a need to create high-performance learners trained with more easily obtained data from different domains. This methodology is referred to as transfer learning. With the transfer learning approach, a neural network trained for a task is reused as a model on a second task (Lumini and Nanni, 2019). Due to the wide application prospects, transfer learning has become a popular and promising area in machine learning. In (Zhuang et al., 2020) there is a survey that systematizes the existing transfer learning researches, as well as summarizes and interprets the mechanisms and the strategies in a comprehensive way, which may help readers have a better understanding of the current research status and ideas. There are different categories of transfer learning. The instance-based transfer learning utilizes similar instances in the source domain to establish the prediction model for the target domain. One of the most representative instance-based transfer learning methods is TradaBoost proposed by Dai et al. (Dai et al., 2007), which is based on AdaBoost. Another one is the kernel mean matching method after the instances are redistributed in reproducing kernel Hilbert space (Huang et al., 2006). The feature-based transfer learning, the main idea of which is to use the source domain instances to generate instances in the target domain. For the transfer between two domains of different characteristics, Nam and Kim (Nam et al., 2017) migrated through two steps of feature selection and connection and achieved relatively good performance. An additional feature-based transfer learning method is proposed by Pan et al. (Pan et al., 2011).

This method proposes a representation through a new learning method, transfer component analysis (TCA), for domain adaptation to map the source and target domains from the original feature space to a potential one. In this paper, we used a special case of transfer learning, called domain adaptation (Pan et al., 2011). In this case, source (external projects) and target (project to be predicted) domains are different while sharing the same learning task. In particular, we considered a model that has learned to classify and predict smells in software projects and is used for classifying smells in a different software project.

### 2.2 Design Smells

This study refers to the design smells reported in (Girish Suryanarayana, 2014), and uses the known Designite software design quality assessment tool (Sharma et al., 2016) to detect these design smells from the software systems. The design smells considered in this study are the following:

- **Multifaced Abstraction (MA):** It arises when more responsibilities are designated to the same abstraction.
- **Unutilized Abstraction (UNA):** It indicates that there is an abstraction that is (i) unused, (ii) not directly used or, (iii) not reachable. This smell can be of two types, respectively named Unreferenced Abstractions and Orphan Abstractions. The former are unused concrete classes. The latter are stand-alone interfaces/abstract classes without any subtypes.
- **Unnecessary Abstraction (UA):** It arises when in the software design unnecessary abstraction is introduced. It causes the violation of the abstract principle asserting that the abstract entities should have only one responsibility and this responsibility should have high importance.
- **Imperative Abstraction (IA):** It indicates that an operation is transformed into a class. However, this smell looks like a class with a unique method.
- **Deficient Encapsulation (DE):** It occurs when the abstraction encapsulation is not sufficient. There are two possible types: lenient and vulnerable encapsulation. Lenient encapsulation occurs when a member of abstraction can access in a more permissive way than required. The latter indicates the vulnerability of an abstraction state for misuse or corruption (implementation details are not more adequately protected).
- **Unexploited Encapsulation (UE):** It arises when the explicit type checks are used by client codes

while the variation in type already encapsulated within a hierarchy should be exploited.

- **Broken Modularization (BM):** It indicates that data or methods that should be into a single abstraction are distributed along with several abstractions.
- **Cyclically-dependent Modularization (CMD):** It arises when two or more class-level abstractions are directly or indirectly dependent on each other. In this case, tight coupling is created between these abstractions.

### 3 RELATED WORK

Several detection approaches and tools (Imran, 2019; Alkharabsheh et al., 2018) for design smells are recently proposed. Mainly, these approaches allow detecting design smells (and the consequent design problems) once they already exist in the code. Other studies explore the adoption of machine learning smell techniques (Sharma et al., 2019; Al-Shaaby, 2020) as an alternative to traditional design smells detection approaches. In particular, in (Sharma et al., 2019) existing metric-based methods and different deep learning architectures (i.e., Convolution Neural Networks (CNNs), Recurrent Neural Networks (RNNs)) are used to detect one design smell in C code. However, authors in (Al-Shaaby, 2020) highlight that the research in the field of application of machine learning algorithms to detect code smells is quite immature. In this context, authors in (Sharma et al., 2019; Di Nucci et al., 2018) highlight that a limitation of the existing studies is that they never consider the possibility to exploit the availability of tools and data about code smells in a programming language as the training set for machine learning models that address similar problems on other languages. Starting from this idea, in this study, we aim to use the data extracted from software projects with a high amount of design smells for training deep learning models useful to predict design smells in projects with a more limited number of data. To the best of our knowledge, this is the first time that transfer learning is applied to design smell prediction using metric-based supervised learning. Differently from (Sharma et al., 2019) that exploits transfer learning for design smell detection from source code, in this study we propose a metric-based approach introducing both product and process metrics to perform design smell prediction at class and commit level across a corpus of open-source software projects, comparing direct and transfer learning performances. Moreover, this

study adopted a Temporal Convolutional Networks (TCN) approach characterized by casualness in the convolution architecture design (Bai et al., 2018) that make it particularly suitable for our prediction problem where the causal relationships among the metrics evolution over time and design smell presence need to be learned.

## 4 THE TRANSFER LEARNING APPROACH

In this study, classifiers are trained using historical data from different adult systems that have similar properties. The built classifiers are then used to predict design smells on the system under study. Figure 2 depicts the performed transfer learning schema. The upper side of the figure shows the learning transfer activities allowing to improve the transfer of training. Starting from software project historical data, we cleaned the raw dataset by removing incomplete and wrong sampled data sessions and normalizing the attributes (min-max normalization). Then the set of considered features are extracted according to the proposed data extraction process (the proposed feature model and the data extraction process will be described in the following section). The data extraction step generates an integrated dataset collecting design smells data and metrics data. This allows to start of the training of the classifiers (one for each considered design smell) and generates the transfer model. We defined a set of labeled traces  $T = (M, l)$ , where  $M$  is an instance associated with a label  $l \in \{U_1, \dots, U_n\}$ , which represents a design smell introduction. For each  $M$ , the process computes a feature vector  $V_f$  submitted to the model in the training phase. To perform validation during the training step, a 10-fold cross-validation is used (Stone, 1974). When a target software project is evaluated a new data extraction process is executed (lower side of the figure). In this step, the process and product metrics are extracted and sent to the classifier. The classifier takes as input the transfer model and the metrics data to perform the design smell prediction. More details about the proposed TCN classifier will be reported in Section 4.2.

### 4.1 Data Extraction

Figure 1 depicts the data extraction steps required for the training process. First of all, for each considered system, the change (revision) history is gathered commit per commit from the GitHub repository. All the commits log messages so obtained are, then, analyzed

and parsed to report all the changes of the files contained in the code repository. In particular, as reported in the Commits Log Analyzer box, the extracted commits logs are analyzed to obtain the process metrics dataset while the commits' source code is used to perform smell detection and product metrics evaluation (Designite box and CK+JaSoMe box). In this step, the analysis of the source code at each commit has been performed to understand and measure its evolution over time, commit per commit. The product metrics are computed by using JaSoMe<sup>1</sup> (Java Source Metrics) and CK<sup>2</sup> object-oriented metrics extractors. They are both open-source code analyzers that mine internal quality metrics from projects based on source code alone.

To detect the design smells, the Designite<sup>3</sup> tool has been used. It is a software design and quality assessment tool allowing to perform a comprehensive design smells detection and a detailed metrics analysis.

The obtained smell dataset and the product metrics dataset are related through a smell-metrics matching activity. Finally, the process metrics and the product metrics (within their related smells) are integrated and stored in a single data set.

As product metrics, we considered the well-known Chidamber and Kemerer metrics (Chidamber and Kemerer, 1994) and the MOOD Metrics (e Abreu and Carapuça, 1994). While, as process metrics, useful to capture both the kind of changes that are performed to source code elements and the properties of authors and committers performing such change, we considered the following ones defined as follows:

- **Developer Seniority (SEN):** It measures the time, expressed in days, between the first commit and the last commit authored by developer  $d_i$ . Its formula is:

$$\text{SEN}(c_j, d_i) = \text{Cd}(c_j) - \text{Fc}(d_i)$$

where:

$c_j$ : the commit for which we want to evaluate the author's seniority.

$d_i$ : the developer who authored the commit  $c_j$ .

$\text{Cd}(c_j)$ : the date of the commit  $c_j$ .

$\text{Fc}(d_i)$ : the date of the first commit authored in the source code repository by developer  $d_i$ .

- **Owned Commit (OC):** It considers the set of *file owners* as the committers that performed together, on the file, a given percentage (specifically, 50%)

of the total number of commits that occurred on that file. In our context, we tag as owners the set of the developers that collectively performed at least half of the total changes on that file.

- **Number of File Owners (NFOWN):** Given the definitions above, this can be defined, for a file  $f_j$  and a commit  $c_k$ , as the cardinality of the owners of the file  $f_j$  at commit  $c_k$ .
- **Owned File Ratio (OFR):** For a developer  $d_i$ , a file  $f_j$  and a commit  $c_k$ , this is the ratio  $R(d_i, f_j, c_k)$  of changes performed by  $d_i$  with respect to the total changes performed by all developers on file  $f_j$  from the start of the observation period (i.e., in the commits interval  $[c_s, \dots, c_k]$ ).

Additional process metrics describe file or developer properties and can be easily computed using VCS logs. The **Time since the last commit (TSLC)**, for a given file  $f_j$  and a commit  $c_k$ , is the number of days passed since the last commit on  $f_j$ . For a given developer  $d_i$ , the **Commit Frequency (CF)** is the number of commits by month authored by  $d_i$  whereas the **Mean time between commits (MTBC)** is the average time (in days) between all subsequent commits authored by  $d_i$ .

## 4.2 TCN Classifier

This study is based on a TCN architecture. In particular, in the TCN network, we use a hierarchical attention mechanism through the network levels introduced by (Yang et al., 2016) and applied in (Ardimento. et al., 2020), because it helps the standard TCN architecture to learn more efficiently the complex temporal relationships present in multivariate time series of internal quality metrics evolution. According to this architecture, if the network has  $n$  hidden levels, a matrix  $L_i$  is defined, it includes the convolutional activations on each layer  $i$  (with  $i = 0, 1, \dots, n$ ) defined as:

$$L_i = [l_0^i, l_1^i, \dots, l_T^i], L_i \in R^{K \times T},$$

where  $K$  represents the number of filters present in each level.

During the training phase, validation is done through 10-fold cross-validation (Stone, 1974). Furthermore, the evaluation of the trained prediction model is carried out using real data that the network has never seen before, data contained in the test set made of classes, and therefore of smells. In particular, the architecture in question was trained using the cross-entropy loss function (Mannor et al., 2005).

The network was developed using the Python programming language and, in particular, the libraries

<sup>1</sup><https://github.com/rodhilton/jasome>

<sup>2</sup><https://github.com/mauricioaniche/ck>

<sup>3</sup><https://www.designite-tools.com>

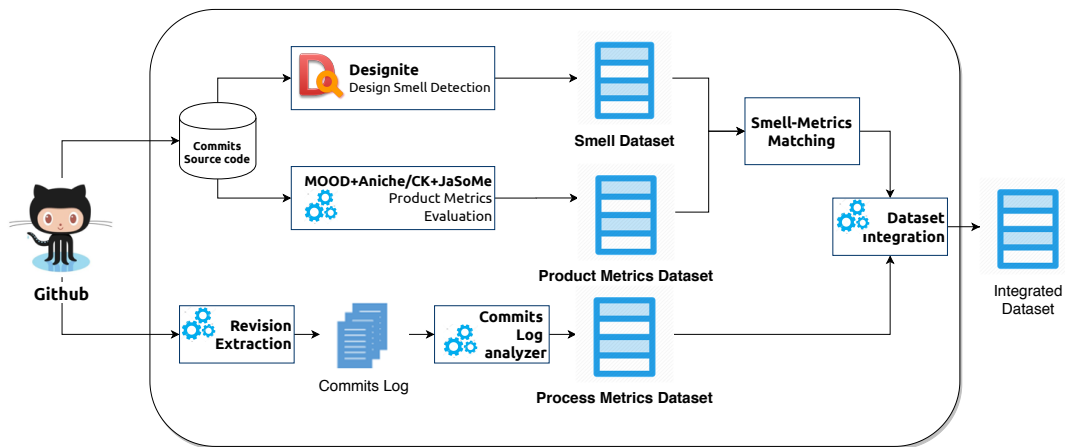


Figure 1: Toolchain used in the data extraction process.

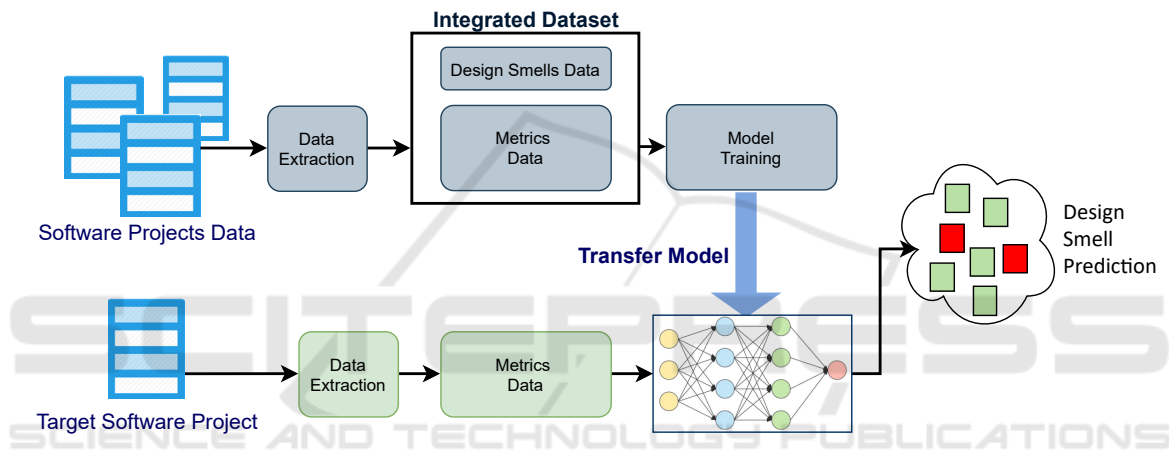


Figure 2: Transfer Learning for Cross-Project Design Smells Prediction.

Tensorflow<sup>4</sup> and Keras<sup>5</sup> were used to implement the designed deep neural network architecture.

## 5 EXPERIMENT DESCRIPTION

The research goal discussed in the introduction can be formulated as the following research question:

**RQ:** *What is the performance of the TCN model to predict design smells occurrences when trained on the proposed features set on data collected across different systems?*

This research question investigates if the performance of the TCN model is effective when trained using Transfer Learning (TL), i.e. trained on a set of software systems different than those on which it is validated. The results are compared to those obtained with Direct Learning considered as the baseline in our

<sup>4</sup><https://www.tensorflow.org>

<sup>5</sup><https://keras.io>

work. This baseline is built on a learning approach based on the TCN model using data extracted from the same system under study. For all these models, hyper-parameters optimization is performed and the comparison is conducted comparing models with the best configurations of each considered model (due to space constraints, we only report information about best parameter configurations for the TCN model).

To evaluate the performance of the prediction, the *F1* score is computed. It is evaluated as the harmonic average of the precision and the recall metrics. Precision is the number of corrected positive results divided by the number of all positive results returned by the models. The recall is the number of corrected positive results divided by the number of all relevant samples.

Also notice that we train a single classifier for each design smell since the relationships among the features and the smells can be quite different and a single neural multinomial classifier would be unable to perform adequately.

Table 1: An overview of the analyzed systems (commits and revisions by system and smell).

Systems →		ZooKeeper	JFreeChart	Jackson DF
From—To dates →		Nov 2007—Mar 2020	Jun 2007—Feb 2020	Jan 2011—Mar 2020
Total commits →		2101	3786	1901
Total revisions →		1264735	403467	253176
Design Smells ↗	Imperative Abstraction (IA)	14088	2097	0
	Multifaceted Abstraction (MA)	1021	1781	0
	Unnecessary Abstraction (UA)	25259	3823	85166
	Unutilized Abstraction (UNA)	796138	285607	134819
	Deficient Encapsulation (DE)	163204	57478	185501
	Unexploited Encapsulation (UE)	5028	0	0
	Broken Modularization (BM)	15418	1774	1040
	Cyclic D. Modularization (CDM)	146644	36591	5244

Table 2: Class imbalance for the analyzed systems.

Class imbalance	ZooKeeper	JFreeChart	Jackson DF
IA	97.77	98.96	-
MA	99.83	88.11	-
UA	96.00	98.10	32.72
UNA	25.89	41.57	6.50
DE	74.19	71.51	46.54
UE	99.20	-	-
BM	97.56	99.12	99.17
CDM	76.81	81.86	95.86

## 5.1 Dataset Construction

In the experiments, a dataset composed of data gathered from three Java open-source projects (ZooKeeper<sup>6</sup>, JFreeChart<sup>7</sup>, Jackson DataFormats<sup>8</sup>) is considered. The selected software projects<sup>9</sup> have these characteristics: i) their programming language is Java; ii) the corresponding repository is not archived and more than 20 releases are available; iii) they differ for application domains, sizes, number of revisions; iv) they are adopted in other studies. The project names are reported in the first row of Table 1. The table also reports for each considered design smell the number of occurrences in the analyzed system.

## 6 DISCUSSION OF RESULTS

In this section, results obtained from direct learning and transfer learning are shown and compared.

Table 3 reports the F1-score obtained, for the best hyper-parameters combination, by the TCN model

<sup>6</sup><https://github.com/apache/zookeeper>

<sup>7</sup><https://github.com/jfree/jfreechart>

<sup>8</sup><https://github.com/FasterXML/jackson-dataformat-xml>

<sup>9</sup>A replication package containing the dataset used for training is provided at <https://bit.ly/3lpEoMq>

Table 3: Best F1-score for each smell and each system obtained by TCN model using direct learning approach.

Design Smells	Software Systems			DL F1
	ZooKeeper	JFreeChart	Jackson DF	
IA	0,12	0,45	-	0,45
MA	0,24	0,33	-	0,33
UA	0,45	0,55	0,97	0,97
UNA	0,94	0,96	0,96	0,96
DE	0,89	0,96	0,95	0,96
UE	0,68	-	-	0,68
BM	0,38	0,56	0,28	0,65
CDM	0,91	0,94	0,72	0,94

Table 4: Best F1-score for each smell and each system obtained by TCN model using transfer learning approach.

Design Smells	Software Systems			TL F1
	ZooKeeper	JFreeChart	Jackson DF	
IA	0,13	0,42	-	0,42
MA	0,22	0,31	-	0,31
UA	0,43	0,53	0,94	0,94
UNA	0,92	0,91	0,93	0,93
DE	0,89	0,91	0,90	0,91
UE	0,76	-	-	0,76
BM	0,33	0,54	0,26	0,54
CDM	0,92	0,94	0,76	0,94

for each considered design smell when direct learning (DR) is used. The table shows that TCN performance strongly depends on the smell. Best performance is obtained by the UNA and DE smells. For some smells, the F1 score can not be evaluated since they are not present in the analyzed systems. To better understand the obtained results, in Table 2, the class imbalance evaluated for each analyzed system is reported. It represents the level of balancing between smelly-not smelly classes in the system and its values range between 0 and 100. A value equal to 0 means absence of unbalance while a value equal to 100 means that there is a total unbalance of smelly and not-smelly classes. Notice that for systems with higher class imbalance, e.g. IA and MA, the obtained performance is low. Table 4 shows the best F1-score for the best hyper-parameters combination, by the TCN model for each considered design smell

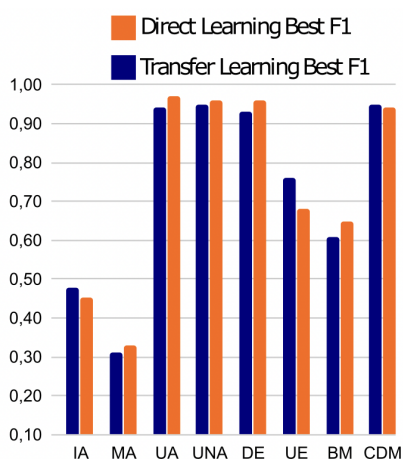


Figure 3: F1-score for TCN with direct learning (DL-Best F1) vs TCN with transfer learning (TL-Best F1).

when transfer learning (TL) is used. The table shows that, also in this case, good performance is obtained for all the systems with a lower class imbalance. It is also worth noting that for class imbalance smaller than 95% TCN F1-score is always over 0.9. This suggests that there is a critical threshold behind which the training performance starts to deteriorate.

A comparison between the F1-score of DL and TL is depicted in Figure 3. It can be observed that for some kind of design smells the obtained values are very satisfying, i.e. for Unused Abstraction, Unnecessary Abstraction, Deficient Encapsulation, Cyclically Dependent Modularization. For some other kind of design smells the F1-score results are not positive, i.e. the Imperative Abstraction and Multifaceted Abstraction design smells. However, the more interesting aspect of these results is that the transfer learning results obtained in any kind of design smell are comparable to the ones obtained by direct learning. This means that the obtained model, especially in the case of design smell with a high F1-score, can be effectively used to predict changes introducing smells even in software projects without a sufficient number of available data. This means that transfer learning can effectively be used when to predict the design smell of new projects with limited history.

## 7 THREATS TO VALIDITY

We analyzed and mitigated the following threats to the validity of our study:

- accuracy of the tools: three widespread tools respectively called Designite, CK, and JaSoMe were used;

- wrong evaluation of the design smells: several experts performed sample checks that include manual inspection and majority voting of evaluated data ensuring consistency;
- meaningfulness of metrics: an accurate process of data gathering was performed.
- generalization of results: well-known software projects with different sizes, dimensions, domains, timeframe, and number of commits were selected.

Finally, our work only considers systems written in Java because the tools used to perform design smells detection and metrics assessment exclusively work only on Java programs. For this reason, we cannot generalize the obtained results to systems written in different languages. Similarly, we did not consider projects belonging to the industrial context.

## 8 CONCLUSIONS

In this study, we proposed and evaluated the adoption of a transfer learning approach to address the just-in-time design smells prediction problem. The empirical results show that when the class is well balanced the prediction model is effective for direct learning and is usable as an alternative with comparable results. Moreover, the results show that transfer learning provides F1 scores very close to the ones obtained by direct learning. To generalize our findings, future work will be devoted to replicating this study on more and larger open-source systems. Another research direction is related to experimenting with novel neural network architectures suitable for multi-variate time series prediction problems.

## REFERENCES

- Al-Shaaby, A. and, A. H. A. M. B. (2020). Smell detection using machine learning techniques: A systematic literature review. *Arab J Sci Eng* 45, 2341-2369.
- Alkharabsheh, K., Crespo, Y., Manso, E., and Taboada, J. A. (2018). Software design smell detection: a systematic mapping study. *Software Quality Journal*, pages 1–80.
- Ardimento., P., Aversano., L., Bernardi., M., and Cimitile., M. (2020). Temporal convolutional networks for just-in-time software defect prediction. In *Proceedings of the 15th International Conference on Software Technologies - ICSOFT*, pages 384–393. INSTICC, SciTePress.
- Aversano., L., Bernardi., M., Cimitile., M., Iammarino., M., and Romanyuk., K. (2020). Investigating on the rela-

- tionships between design smells removals and refactorings. In *Proceedings of the 15th International Conference on Software Technologies - ICSOFT*, pages 212–219. INSTICC, SciTePress.
- Bai, S., Kolter, J. Z., and Koltun, V. (2018). An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *CoRR*, abs/1803.01271.
- Chidamber, S. R. and Kemerer, C. F. (1994). A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493.
- Dai, W., Yang, Q., Xue, G., and Yu, Y. (2007). Boosting for transfer learning. In Ghahramani, Z., editor, *Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007), Corvallis, Oregon, USA, June 20-24, 2007*, volume 227, pages 193–200. ACM.
- Di Nucci, D., Palomba, F., Tamburri, D. A., Serebrenik, A., and De Lucia, A. (2018). Detecting code smells using machine learning techniques: Are we there yet? In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 612–621.
- de Abreu, F. B. and Carapuça, R. (1994). Candidate metrics for object-oriented software within a taxonomy framework. *Journal of Systems and Software*, 26(1):87–96.
- Girish Suryanarayana, Ganesh Samarthayam, T. S. (2014). *Refactoring for Software Design Smells: Managing Technical Debt*. Morgan Kaufmann.
- Huang, J., Smola, A. J., Gretton, A., Borgwardt, K. M., and Schölkopf, B. (2006). Correcting sample selection bias by unlabeled data. In Schölkopf, B., Platt, J. C., and Hofmann, T., editors, *Advances in Neural Information Processing Systems 19: Proceedings of the 2006 Conference, Vancouver, British Columbia, Canada, December 4-7, 2006*, pages 601–608. MIT Press.
- Imran, A. (2019). Design smell detection and analysis for open source java software. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 644–648.
- Kitchenham, B. A., Mendes, E., and Travassos, G. H. (2007). Cross versus within-company cost estimation studies: A systematic review. *IEEE Transactions on Software Engineering*, 33(5):316–329.
- Lumini, A. and Nanni, L. (2019). Deep learning and transfer learning features for plankton classification. *Ecological informatics*, 51:33–43.
- Mannor, S., Peleg, D., and Rubinstein, R. (2005). The cross entropy method for classification. In *Proceedings of the 22Nd International Conference on Machine Learning, ICML '05*, pages 561–568, New York, NY, USA. ACM.
- Nam, J., Fu, W., Kim, S., Menzies, T., and Tan, L. (2017). Heterogeneous defect prediction. *IEEE Transactions on Software Engineering*, 44(9):874–896.
- Pan, S., Tsang, I., Kwok, J., and Yang, Q. (2011). Domain adaptation via transfer component analysis. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 22:199–210.
- Sharma, T., Efstathiou, V., Louridas, P., and Spinellis, D. (2019). On the feasibility of transfer-learning code smells using deep learning. *CoRR*, abs/1904.03031.
- Sharma, T., Mishra, P., and Tiwari, R. (2016). Designite - a software design quality assessment tool. In *2016 IEEE/ACM 1st International Workshop on Bringing Architectural Design Thinking Into Developers' Daily Activities (BRIDGE)*, pages 1–4.
- Stone, M. (1974). Cross-validatory choice and assessment of statistical predictions. *Roy. Stat. Soc.*, 36:111–147.
- Suryanarayana, G., Samarthayam, G., and Sharma, T. (2015). Chapter 2 - design smells. In Suryanarayana, G., Samarthayam, G., and Sharma, T., editors, *Refactoring for Software Design Smells*, pages 9 – 19. Morgan Kaufmann, Boston.
- Yang, Z., Yang, D., Dyer, C., He, X., Smola, A., and Hovy, E. (2016). Hierarchical attention networks for document classification. In *Proceedings of NAACL-HLT 2016*, pages 1480–1489, San Diego, California. Association for Computational Linguistics.
- Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., Xiong, H., and He, Q. (2020). A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76.