Convolutional Neural Network programs meta-modeling for automatic detection of design smells

Aurel Ikama[1], Foutse Khomh[1], Heng Li[1]
[1] *Dep. of Computer and Software Engineering, Polytechnique Montréal, Montréal*

***Abstract*—TODO: To writte after the paper is finished**
***Keywords*—** Deep Learning, Convolutional Neural Network, Design Smells, Meta-Modeling, Software Engineering

I. Introduction

L'apprentissage profond est un sous ensemble du machine learning qui utilise des réseaux de neurones artificiels pour apprendre des données non structurées. L'apprentissage profond est devenu très populaire ces dernières années grâce à ses performances dans plusieurs domaines comme la reconnaissance d'image, la reconnaissance vocale, la traduction automatique, etc. Les réseaux de neurones artificiels sont des modèles mathématiques qui sont inspirés du fonctionnement du cerveau humain. Ils sont composés de plusieurs couches de neurones qui sont connectés entre eux. Chaque neurone est composé d'une fonction d'activation qui permet de calculer la sortie du neurone en fonction de ses entrées. Il existe plusieurs types d'architecture de réseaux de neurones artificiels. Ses architecture ont été développées pour répondre à des problèmes spécifiques tels que l'architecture de convolution pour la reconnaissance d'image, les réseaux de neurones récurrents pour la reconnaissance vocale, etc. Dans ce papier, nous nous intéressons à l'architecture de convolution car elle permet de traiter un large ensemble de problèmes comme la reconnaissance d'image, la reconnaissance de texte, la reconnaissance de séquence, etc. Par ailleurs l'architecture de convolution fait partie des architectures feed-forward qui sont une classe de réseaux de neurones artificiels qui sont composés de plusieurs couches de neurones et qui ne contiennent pas de cycles. Les réseaux de neurones feed-forward sont les plus utilisés dans l'apprentissage profond.

Nous parlerons dans ce papier de programmes d'apprentissage profond pour désigner les programmes contenant les réseaux de neurones. Tout comme n'importe quel programme, les programmes d'apprentissage profond peuvent contenir des défauts. Nous nous intéressons dans ce papier aux défauts de conception car se sont des défauts introduits tôt dans le cycle de développement du logiciel et ils peuvent avoir un impact négatif conséquent sur la performance et la qualité du logiciel. En effet les défauts de conception sont des défauts qui sont introduits par le développeur lors de la phase de conception du logiciel.

- Définition des défauts de conception
- Définit les défauts de conception dans les programmes d'apprentissage profond
- Définition différence entre odeurs, bugs, erreurs et défauts

Méta modélisation des programmes d'apprentissage profond

RQ1: Peut-on détecter les odeur de conception dans les programmes d'apprentissage profond CNN avec la méta-modélisation?

RQ2: Quels sont les odeurs de conception les plus répandu dans les programmes d'apprentissage profond CNN?

RQ3: Exist-il des lien entre les odeurs de conception dans les programmes?

II. Background

In this section, we present the related work. We first present the related work on the detection of design smells in traditional software. Then, we present the related work on the detection of design smells in deep learning software. Finally, we present the related work on the meta-modeling of deep learning software.

A. Detection of design smells in traditional software

design smells are design flaws that are introduced by the developer during the design phase of the software development life cycle. They are considered as a threat to the quality of the software because they can lead to maintainability issues such as code smells, bugs, and performance issues. Several studies have been conducted to detect design smells in traditional software. In this section, we present the related work on the detection of design smells in traditional software.

1) Detection of design smells using static analysis: Static analysis is a technique that is used to analyze the source code of a software without executing it. It is used to detect design smells in software.

Detection of design smells using metrics. Metrics are used to detect design smells in software. Several studies have been conducted to detect design smells using metrics. For example, Marinescu et al. [**?**] proposed a technique to detect design smells using metrics. They used the CK metrics suite [**?**] to detect design smells in software. They used the CK metrics suite to compute the metrics of the software and then they used a threshold to detect the design smells. They used the following thresholds: WMC $\geq$ 47, DIT $\geq$ 6, NOC $\geq$ 7, CBO $\geq$ 6, RFC $\geq$ 47, LCOM $\geq$ 1, and NPM $\geq$ 21. They used the thresholds to detect the design smells in the software.

Detection of design smells using machine learning. Machine learning is a technique that is used to detect design smells in software. Several studies have been conducted to detect design smells using machine learning. For example, Palomba et al. [**?**] proposed a technique to detect design smells using machine learning. They used the CK metrics suite [**?**] to compute the metrics of the software and then they used machine learning to detect the design smells. They used the following metrics: WMC, DIT, NOC, CBO, RFC, LCOM, and NPM. They used the following machine learning algorithms: Decision Tree, Random Forest, Naive Bayes, Support Vector Machine, and Logistic Regression. They used the machine learning algorithms to detect the design smells in the software.

2) Detection of design smells using dynamic analysis: Dynamic analysis is a technique that is used to analyze the behavior of a software during its execution. It is used to detect design smells in software.

Detection of design smells using metrics. Metrics are used to detect design smells in software. Several studies have been conducted to detect design smells using metrics. For example, Marinescu et al. [**?**] proposed a technique to detect design smells using metrics. They used the CK metrics suite [**?**] to detect design smells in software. They used the CK metrics suite to compute the metrics of the software and then they used a threshold to detect the design smells. They used the following thresholds: WMC $\geq$ 47, DIT $\geq$ 6, NOC $\geq$ 7, CBO $\geq$ 6, RFC $\geq$ 47, LCOM $\geq$ 1, and NPM $\geq$ 21. They used the thresholds to detect the design smells in the software.

Detection of design smells using machine learning. Machine learning is a technique that is used to detect design smells in software.

Several studies have been conducted to detect design smells using machine learning. For example, Palomba et al. [**?**] proposed a technique to detect design smells using machine learning. They used the CK metrics suite [**?**] to compute the metrics of the software and then they used machine learning to detect the design smells. They used the following metrics: WMC, DIT, NOC, CBO, RFC, LCOM, and NPM. They used the following machine learning algorithms: Decision Tree, Random Forest, Naive Bayes, Support Vector Machine, and Logistic Regression. They used the machine learning algorithms to detect the design smells in the software.

B. Detection of design smells in deep learning software

Deep learning software is a type of software that is used to train deep learning models. It is used to detect design smells in deep learning software.

Detection of design smells using metrics. Metrics are used to detect design smells in software. Several studies have been conducted to detect design smells using metrics. For example, Marinescu et al. [**?**] proposed a technique to detect design smells using metrics. They used the CK metrics suite [**?**] to detect design smells in software. They used the CK metrics suite to compute the metrics of the software and then they used a threshold to detect the design smells. They used the following thresholds: WMC $\geq$ 47, DIT $\geq$ 6, NOC $\geq$ 7, CBO $\geq$ 6, RFC $\geq$ 47, LCOM $\geq$ 1, and NPM $\geq$ 21. They used the thresholds to detect the design smells in the software.

Detection of design smells using machine learning. Machine learning is a technique that is used to detect design smells in software. Several studies have been conducted to detect design smells using machine learning. For example, Palomba et al. [**?**] proposed a technique to detect design smells using machine learning. They used the CK metrics suite [**?**] to compute the metrics of the software and then they used machine learning to detect the design smells. They used the following metrics: WMC, DIT, NOC, CBO, RFC, LCOM, and NPM. They used the following machine learning algorithms: Decision Tree, Random Forest, Naive Bayes, Support Vector Machine, and Logistic Regression. They used the machine learning algorithms to detect the design smells in the software.

C. Meta-modeling of deep learning software

Meta-modeling is a technique that is used to model the software. It is used to model the deep learning software. Several studies have been conducted to model the deep learning software. In this section, we present the related work on the meta-modeling of deep learning software.

Meta-modeling of deep learning software using metrics. Metrics are used to model the deep learning software. Several studies have been conducted to model the deep learning software using metrics. For example, Marinescu et al. [**?**] proposed a technique to model the

Meta-modeling of deep learning software using machine learning. Machine learning is a technique that is used to model the deep learning software.

Several studies have been conducted to model the deep learning software using machine learning. For example, Palomba et al. [**?**] proposed a technique to model the deep learning software using machine learning. They used the CK metrics suite [**?**] to compute the metrics

III. Study Design

In this sub-section, we describe the details of the data collection and processing approach followed to answer our different research questions.

A. Data Collection

Le développement du système de détection d'odeurs de conception s'est fait à partir des exemples de code de programmes d'apprentissage profond. Ces exemples représentent des programmes d'apprentissage profond dans lesquels on y trouve des odeurs de conceptions. Ses exemples ont été collectés par Amin et al. dans le cadre de leur étude empirique sur les odeurs de conception dans les programmes d'apprentissage profond. En effet, dans cette étude, ils ont présenté sept exemples de programmes d'apprentissage profond contenant les ordeurs de conceptions énuméré dans l'introduction I. Ces exemples de code source ont été collectés à partir des plateformes StackOverflow et Github.

Le système de détection d'odeurs de conception ainsi développé a servi sur un ensemble de programmes d'apprentissage profond collectés à partir de la plateforme Github. Ces programmes d'apprentissage profond ont été collectés selon un processus bien défini. En effet, nous avons utilisé l'API de Github et plus précisement les requêtes de recherche de type *search code*. Cette requête permet de rechercher des fichiers dans les dépôts Github contenant des mots clés spécifiques définis dans notre système. Étant donné que notre papier se concentre uniquement sur les libraires *Keras*, *Tensorflow* et *Pytorch*, nous avons utilisé les mots clés présentés dans le tableau I. Ces mots clés représentent les modules et les fonctions de ces trois librairies.

Table I: *Liste des mots clés utilisés pour la recherche de programmes d'apprentissage profond dans les dépôts Github.*

Mots clés	**Librairie**
keras.layers	Keras
keras.layers.convolutional	Keras
AveragePooling2D	Keras/Tensorflow
MaxPooling2D	Keras/Tensorflow
tensorflow.keras.layers	Tensorflow
Conv2D	Keras/Tensorflow
Convolution2D	Keras/Tensorflow
BatchNormalization	Keras/Tensorflow
import torch	Pytorch
import torchvision.models	Pytorch
torch.nn.Sequential	Pytorch
torch.nn.Conv2d	Pytorch
torch.nn.BatchNorm2d	Pytorch
torch.nn.MaxPool2d	Pytorch

La requête de recherche de type *search code* retourne un ensemble d'informations sur le repository et le fichier contenant les mots clés recherchés. Cette collecte retourne un ensemble de données brute de 9572 repositories Github diférents. Nous sélectionnons ensuite aléatoirement un sous ensemble de 10% de notre ensemble de données brutes, soit 958 repositories Github afin d'optimiser l'étape de filtrage manuel ci-après.

À partir de ce sous ensemble, nous procédons ensuite à un filtrage des répertoires selon les critères suivants: (1) nombre de

commit, (2) notre d'étoile, (3) nombre de fork, (4) dernière date du commit, (5) nombre de contributeurs. Ce filtrage nous permet de ne garder que les répertoires qui sont les plus populaires et qui sont les plus actifs. Nous avons ensuite procédé à un filtrage manuel des répertoires en éliminant les projets qui ne sont pas des programmes d'apprentissage profond ou ne pas pertinente pour notre étude (n'utilise pas les librairies choisies, ou n'implémente pas de réseau de neurones). De ce fait nous nous sommes retrouvé avec 500 projets sur lesquels nous allons appliquer notre système de détection d'odeurs de conception.
Le schema **??** présente le processus de collecte des programmes d'apprentissage profond.

B. Data Processing

Le processus de collecte des programmes d'apprentissage profond nous a permis d'avoir un ensemble de programmes d'apprentissage profond sur lesquels nous allons appliquer notre système de détection d'odeurs de conception. Cependant, il est important de noter que les programmes d'apprentissage profond sont dans notre cas des programmes uniquement écrits en langage Python. Il est donc essentiel de noter que, le système de détection d'odeurs de conception développé dans ce papier est un système qui est capable de détecter les odeurs de conception dans les programmes d'apprentissage profond écrits en langage Python.

Afin de répondre à la question de recherche *RQ1*, nous avons développé un système de détection d'odeurs de conception dans les programmes d'apprentissage profond. Ce système se décompose en trois parties. La première partie est la partie relative à la méta-modélisation des programmes d'apprentissage profond. La deuxième partie est la partie relative à la détection des odeurs de conception dans les programmes d'apprentissage profond. La troisième partie est la partie relative à l'analyse des résultats de la détection des odeurs de conception dans les programmes d'apprentissage profond.

1) Meta-modélisation des programmes d'apprentissage profond: La méta-modélisation des programmes d'apprentissage profond est la première étape du processus de détection des odeurs de conception dans les programmes d'apprentissage profond. Cette étape consiste à transformer le code source des programmes d'apprentissage profond collecté en un modèle intermédiaire qui sera utilisé pour la détection des odeurs de conception. Ce modèle intermédiaire est un modèle de type *Famix Abstract Syntax Tree* (FAST). Le modèle FAST est un modèle qui est utilisé pour représenter les programmes sous forme d'arbre syntaxique. Il hérite de la représentation abstraite de codes sources *Famix* dévéloppé dans le langage de programmation *Pharo*. Avec *Famix*, le code source est représenté sous forme d'objet et selon un méta-modèle défini pour symplifier l'analyse et la représentation de programmes complexes. Un modèle *Famix* est une représentation agnostique en terme de langage du code source d'origine et embarque des fonctions qui permettent entre autres sa navigation et sa transformation.

La transformation du code source collecté en modèle FAST se fait comme suit. Chaque projet collecté est cloné dans un répertoire local. Son code source est ensuite parseé à l'aide de la librairie *ast.py* de Python. Cette librairie permet de transformer le code source en un arbre syntaxique sous forme Json. Le Json obtenu est ensuite transformé en un modèle FAST à l'aide d'un importateur développé sous le patron de conception visiteur. Les visiteurs sont des objets qui permettent de parcourir un arbre syntaxique et d'effectuer des actions sur les noeuds de l'arbre. Dans notre cas, nous avons développé un visiteur qui nous permet de parcourir l'arbre syntaxique (Json) et de transformer chaque noeud de cet arbre en un objet FAST.

Le schema **??** présente le processus de méta-modélisation des programmes d'apprentissage profond.

2) Détection des odeurs de conception: Le modèle FAST obtenu à partir de la méta-modélisation des programmes d'apprentissage profond est ensuite utilisé pour la détection des odeurs de conception. Cette étape consiste à parcourir le modèle FAST et à appliquer les règles de détection des odeurs de conception. Les règles de détection des odeurs de conception sont des règles qui permettent de détecter les odeurs de conception dans les programmes d'apprentissage profond. Ces règles sont définies dans le langage de programmation *Pharo*.
Une règle ou plusieurs règles peuvent être appliquées sur un noeud de l'arbre syntaxique afin de détecter une odeur de conception. Le schema **??** est un exemple de règle permettant de détecter l'odeur de conception numéro 5 - *Non-dominating down-sampling*.
Le schema **??** présente le processus de détection des odeurs de conception dans les programmes d'apprentissage profond.

3) Analyse des résultats de la détection des odeurs de conception: Les résultats de la détection des odeurs de conception de chaque repository sont stockés sous forme *csv*. Pour chaque repository nous enregistrons le nom de chaque odeur de conception détectée, le nombre de fois que cette odeur est détectée dans un fichier du repository et le chemin de ce fichier. Ces résultats permettent de répondre aux questions de recherche *RQ2* et *RQ3*.
On calcul ensuite la répartition des odeurs de conception dans chaque repository en divisant le nombre de fois qu'une odeur de conception est détectée par le nombre total d'odeurs de conception détectées dans le repository. Cette répartition permet de répondre à la question de recherche *RQ2*.
On calcul également la répartition des odeurs de conception dans l'ensemble des 500 repositories en divisant le nombre de fois qu'une odeur de conception est détectée par le nombre total d'odeurs de conception détectées dans l'ensemble des 500 repositories. Cette répartition permet aussi de répondre à la question de recherche *RQ2*.
On calcul ensuite la matrice de corrélation entre les odeurs de conception dans l'ensemble des 500 repositories. Cette matrice permet de répondre à la question de recherche *RQ3*. Une matrice de corrélation est une matrice qui permet de calculer la corrélation entre deux variables.

Un coeficient de corrélation est calculé à l'aide de la formule suivante: $r = \frac{cov(X,Y)}{\sigma_X \sigma_Y}$ où r est le coeficient de corrélation, $cov(X,Y)$ est la covariance entre les deux variables X et Y, σ_X est l'écart type de la variable X et σ_Y est l'écart type de la variable Y. Ce coeficient de corrélation est compris entre -1 et 1. On parle de correlation positive lorsque le coeficient de corrélation est compris entre 0 et 1. Dans ce cas plus il est proche de 1, plus les deux variables sont corrélées positivement, et plus il est proche de 0, moins les deux variables sont corrélées. On parle de correlation négatif lorsque le coeficient de corrélation est compris entre -1 et 0. Dans ce cas, plus le coeficient de corrélation est proche de -1, plus les deux variables sont corrélées négativement, et plus il est proche de 0, moins les deux variables sont corrélées.

Dans notre cas, les variables sont les odeurs de conception. La corrélation entre deux odeurs de conception est calculée en divisant le nombre de fois que les deux odeurs de conception sont détectées ensemble par le nombre total de fois que les deux odeurs de conception sont détectées. Cette corrélation est calculée pour chaque paire d'odeurs de conception. Plus elle est proche de 1, plus les deux odeurs de conception sont corrélées, et plus elle est proche de 0, moins les deux odeurs de conception sont corrélées.

Une corrélation positive signifie que les deux odeurs de conception sont souvent détectées ensemble. Et une corrélation négative signifie que les deux odeurs de conception sont rarement

détectées ensemble. Enfin une corrélation nulle signifie que les deux odeurs de conception ne sont jamais détectées ensemble.

Le schema **??** présente le processus d'analyse des résultats de la détection des odeurs de conception dans les programmes d'apprentissage profond.

C. Replication Package

Le code source du parse code source en arbre de Syntax *Json* est sur Github à l'adresse https://github.com/aurpur/parserPythonToJson et du système de détection d'odeurs de conception dans les programmes d'apprentissage profond est disponible sur Github à l'adresse https://github.com/aurpur/famixPythonImporter. Le code source du système de collection de données est disponible sur Github à l'adresse https://github.com/aurpur/ms-github-data-collection.

IV. Case Study Results

Dans cette section nous présentons les résultats de notre étude. Nous présentons d'abord les résultats relative à la détection des odeurs de conception dans les programmes CNN. Puis, nous présentons les résultats relative de l'analyse des résultats de la détection des odeurs de conception dans les programmes CNN.

A. Détection des odeurs de conception dans les programmes CNN

Ici nous présenterons les chiffres sur la détection des odeurs de conception dans l'ensemble des repositories tel que la répartition des odeurs de conception dans le dataset (Le nombre de repositories par odeurs de conception), ou la profondeur (Nombre d'occurence d'une odeur de conception dans un repository).

Cela soutiendra l'hypothèse selon laquelle les odeurs de conception peuvent être détectées sur les programmes CNN à travers un model FAST dans Pharo. Et de ce fait, on peut donc détecter les odeurs de conception à l'aide de la méta-modélisation.

B. Analyse des résultats de la détection des odeurs de conception dans les programmes CNN

Ici nous présenterons les chiffres sur l'analyse des résultats de la détection des odeurs de conception dans les programmes CNN. Le but est de présenter le classement des odeurs de conception dans les programmes CNN, et de présenter les différentes liens entre elles.

Cela soutiendra l'hypothèse selon laquelle il y a des odeurs de conception qui sont plus répandu que d'autres dans les projets CNN et que certaines odeurs de conception sont liées entre elles.

V. Discussion

Dans cette section, nous discutons les résultats de notre étude et répondons à nos questions de recherche. Nous discutons également les conséquences de nos résultats dans le contexte de la recherche et du développement de logiciels.

A. RQ1: Peut-on détecter les odeur de conception dans les programmes d'apprentissage profond CNN avec la méta-modélisation?

Ici nous discuterons des résultats de notre première question de recherche. Nous discuterons de la technique de détection des odeurs de conception proposée et de sa performance. Nous calculerons l'exactitude, la précision et le rappel de la technique proposée. On discutera également du temps d'exécution et de la mémoire utilisée par la technique proposée.

Nous parlerons de l'objectif derrière la question (Motivation), rappelerons la méthode utilisée et les trouvailles.

B. RQ2: Quels sont les odeurs de conception les plus répandu dans les programmes d'apprentissage profond CNN?

Ici nous discuterons des résultats de notre deuxième question de recherche. Nous discuterons de la répartition et la profondeur des odeurs de conception dans l'ensemble de données.

Nous parlerons de l'objectif derrière la question (Motivation), rappelerons la méthode utilisée et les trouvailles.

C. RQ3: Exist-il des lien entre les odeurs de conception dans les programmes?

Ici nous discuterons des relations entre les odeurs de conception dans les programmes d'apprentissage profond.

Nous parlerons de l'objectif derrière la question (Motivation), rappelerons la méthode utilisée et les trouvailles.

D. Implications

Ici nous parlerons de l'impact de notre étude sur la recherche et le développement de logiciels. Nous parlerons aussi de l'impact de nos résultats.

VI. Threats to validity

Notre étude est sujet de plusieurs menaces à la validité. Dans cette section nous présonterons ces menaces et les stratégies que nous avons utilisé pour les mitiger.

Premièrement, notre étude est sujet de la menace de validité de construction. En effet le modèle FAST généré par notre approche pourrait ne pas être correct et le moèle pourrai ne ma être conforme au méta-modèle défini ou encore mal représenter le programme d'apprentissage profond en entrée. Pour mitiger cette menace, nous avons mise en place des tests unitaires pour chaque méthode visiteur. Ces tests unitaires nous permettent de vérifier que les méthodes implémentées dans l'importateur son correctement implémentées et que le modèle FAST généré est conforme au méta-modèle défini. De plus, nous avons implémenté pour un ensemble de données de tests (des exemples synthétiques) des tests fonctionnels pour vérifier que le modèle FAST généré représente bien le programme d'apprentissage profond en entrée.

Deuxièmement, notre étude est sujet de la menace de validité interne. Il subsiste un risque que les conclusions obtenues ne soient pas causées par les programmes en entrée. Pour mitiger cette menace, nous avons en plus de tester notre système sur des exemples synthétiques, nous avons utilisé des projets réels collectés dans Github pour valider le fonctionnement de notre système.

Troisièmement, notre étude est sujet de la menace de validité externe. Il y a un risque que les résultats obtenus ne soient pas généralisables. Pour mitiger cette menace, nous avons collecté aléatoirement des projets dans Github afin de représenter différente application de l'architecture CNN.

Quatrièmement, notre étude est sujet de la menace de validité de fiabilité. Il y a enfin un risque que les résultats obtenus ne soient pas reproductibles. Pour mitiger cette menace, nous avons en plus d'utilisé des projets de différentes applications de l'architecture CNN, utilisé des projets implémentées dans les trois librairies les plus utilisées dans l'industrie (Tensorflow, Pytorch et Keras).

VII. Limitations and Future Work

1) Limitations: Our study has several limitations. First, our study is limited to the CNN programs. Second, our study is limited to the CNN programs. Third, our study is limited to the CNN programs. Fourth, our study is limited to the CNN programs. Fifth, our study is limited to the CNN programs. Sixth, our study is limited to the CNN programs. Seventh, our study is limited to the CNN programs.

Eighth, our study is limited to the CNN programs. Ninth, our study is limited to the CNN programs. Tenth, our study is limited to the CNN programs. Eleventh, our study is limited to the CNN programs. Twelfth, our study is limited to the CNN programs. Thirteenth, our study is limited to the CNN programs. Fourteenth, our study is limited to the CNN programs. Fifteenth, our study is limited to the CNN programs. Sixteenth, our study is limited to the CNN programs. Seventeenth, our study is limited to the CNN programs. Eighteenth, our study is limited to the CNN programs. Nineteenth, our study is limited to the CNN programs. Twentieth, our study is limited to the CNN programs. Twenty-first, our study is limited to the CNN programs. Twenty-second, our study is limited to the CNN programs.

2) Future Work: Future work will focus on the following directions. First, future work will focus on the following directions. Second, future work will focus on the following directions. Third, future work will focus on the following directions.

VIII. Conclusion

[1]

References

[1] D. Jhon, "test," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2021, pp. 1082–1086.