SPI-GCN: A Simple Permutation-Invariant Graph Convolutional Network

Asma Atamna, Nataliya Sokolovska, Jean-Claude Crivello

SPI-GCN: A Simple Permutation-Invariant Graph Convolutional Network*

Asma Atamna[1], Nataliya Sokolovska[2], and Jean-Claude Crivello[1]

[1] ICMPE (UMR 7182), CNRS, University of Paris-Est, Thiais, France
name@icmpe.cnrs.fr
[2] NutriOmics, INSERM, Sorbonne University, Paris, France
nataliya.sokolovska@sorbonne-universite.fr

Abstract. A wide range of machine learning problems involve handling graph-structured data. Existing machine learning approaches for graphs, however, often imply computing expensive graph similarity measures, preprocessing input graphs, or explicitly ordering graph nodes. In this work, we present a novel and simple convolutional neural network architecture for supervised learning on graphs that is provably invariant to node permutation. The proposed architecture operates directly on arbitrary graphs and performs no node sorting. It also uses a simple multilayer perceptron for prediction as opposed to conventional convolution layers commonly used in other deep learning approaches for graphs. Despite its simplicity, our architecture is competitive with state-of-the-art graph kernels and existing graph neural networks on benchmark graph classification data sets. Our approach clearly outperforms other deep learning algorithms for graphs on multiple multiclass classification tasks. We also evaluate our approach on a real-world original application in materials science, on which we achieve extremely reasonable results.

Keywords: Graph neural networks · Graph convolution · Graph classification.

1 Introduction

Many real-world data present an inherent structure and can be modelled as sequences, graphs, or hypergraphs. Machine learning applications involving such structured data include natural language processing [8], modelling of DNA and RNA sequences [28], molecular property prediction [12], and link prediction in citation and social graphs [2]. Graph-structured data in particular are very common in practice and are at the heart of this work.

We distinguish two main classification problems on graphs: *node* classification and *graph* classification. In this work, we consider the problem of graph classification, that is, given a set $\mathcal{G} = \{G_i\}_{i=1}^m$ of arbitrary graphs G_i and their respective labels $\{y_i\}_{i=1}^m$, where $y_i \in \{1, \ldots, C\}$ and $C \geq 2$ is the number of

* Supported by the Emergence@INC-2018 program of the French National Center for Scientific Research (CNRS).

classes, we aim at finding a mapping $f_\theta(G) : \mathcal{G} \to \{1, \ldots, C\}$, where θ denotes the parameters to optimize.

A variety of graph kernels [16,26,25] have been proposed in the literature to tackle the aforementioned classification problem. The idea is to define similarity measures (i.e. kernels) on graphs that can then be used by classical kernel methods—such as support vector machines (SVMs)—to perform classification.

The advent of deep learning approaches in the past decades and their spectacular success on a variety of tasks, such as image processing, have led to the emergence of multiple deep architectures for graphs. Graph convolutional networks in particular extend convolutional neural networks (CNNs), traditionally defined for regular grids, to graphs by trying to generalize the concepts of "convolution" and "pooling" to graph-structured data. Despite their efficiency, deep learning approaches for graphs usually have very elaborate architectures that render the interpretation of the proposed models difficult [35].

In this paper, we present a novel and simple graph convolutional network for graph classification that is inspired by the work of [21] on semi-supervised node classification. Our architecture supports input graphs of varying sizes and structures represented as adjacency and node feature matrices, and can be applied directly to any graph classification task (e.g. bioinformatics and social graphs) as illustrated in our experiments. We are motivated in particular by a challenging application in materials science, which consists in assessing the *stability* of new chemical compounds in order to store hydrogen efficiently. This tedious task is traditionally tackled through expensive DFT (density functional theory) calculations [22].

Our contributions in this work are multifold:

- We propose a novel end-to-end graph convolutional network for graph classification that is able to process arbitrary graphs directly without any preprocessing;
- The architecture of our Simple Permutation-Invariant Graph Convolutional Network (SPI-GCN) is simple in that (i) no node sorting is required and (ii) the new graph features extracted after the graph convolution are fed to a simple multilayer perceptron (MLP) to perform classification;
- SPI-GCN is *invariant* to node permutation—i.e. graph isomorphism (GI), which guarantees that the same output is returned for equivalent graphs;
- We demonstrate through numerical experiments on benchmark graph data sets that SPI-GCN is competitive with state-of-the-art approaches. It also clearly outperforms similar deep learning methods on multiclass classification problems;
- We also test SPI-GCN on a rich original data set gathered and managed by the ICMPE[3] and dedicated to the generation of new stable crystal structures;
- Our PyTorch implementation of SPI-GCN and the data sets are publicly available at https://github.com/asmaatamna/SPI-GCN.

This paper is organized as follows. Section 2 discusses the related work. We introduce some graph theory concepts and notations in Section 3. In Section 4, we

[3] East Paris Institute of Chemistry and Materials Science, France.

introduce our graph convolutional network, SPI-GCN. Our experimental results are presented in Section 5. Finally, a discussion and concluding remarks are provided in Section 6.

2 Related Work

There are two main supervised machine learning approaches for graph-structured data: graph kernels and graph neural networks.

Graph kernels [16,25] define a similarity measure on graphs thereby allowing the application of kernel machines, such as SVMs, to graph classification. To measure the similarity between two graphs, most graph kernels proceed by decomposing each graph into substructures, e.g. subtrees [39], graphlets [40], or shortest paths [45,3], then comparing these substrucutres pairwise. Graph decomposition can be very expensive for large graphs, and research has focused on designing tractable kernels. An effective class of graph kernels are the Weisfeiler-Lehman (WL) kernels [39] that implement a feature extraction mechanism based on the WL algorithm [47] for graph isomorphism test; the proposed kernels, however, only support discrete features. In [48], a general framework is presented where a deep neural network is used to learn latent representations of the substructures used by different graph kernels, as a way to leverage the dependencies between substructures. Recent works on graph kernels [23,29] discuss the benefit of comparing more global graph substructures. Graph kernels have been the state-of-the-art in graph classification; their main drawback, however, is their computational inefficiency. In particular, the training complexity of graph kernels is at least quadratic in the number of graphs [39].

Graph neural networks were first introduced in [15,36]. The so-called Graph Neural Network (GNN) presented in [36] is a recurrent architecture that learns latent node representations for each node using its neighbors' information until a fixed point is reached. More recent deep learning approaches for graphs aim at generalizing conventional CNNs from regular grids to irregular graph domains; they are commonly referred to as graph convolutional networks (GCNs) and are usually classified into two categories in the literature: *spectral* GCNs, rooted in graph signal processing and where the graph convolution is defined as a function of the graph's Laplacian matrix, and *spatial* GCNs where the graph convolution consists in collecting local information from a node's neighborhood. A pioneering spectral approach is presented in [6], then extended in [17] by the introduction of a graph estimation procedure. In [43], a graph attention network (GAT) model for node classification is proposed with a convolution operator that implicitly assigns different weights to neighboring nodes. A related recent work is presented in [41], where the proposed spatial GCN also assigns different weights to neighboring nodes by exploiting edge features as a way to collect more refined structural information. [10] presents a spectral approach where localized convolution filters are defined as Chebyshev polynomials of the diagonal eigenvalues matrix associated to the graph's Laplacian, thereby reducing the $\mathcal{O}(n^2)$ cost of spectral convolutions on graphs. [21] introduces a first-order approxima-

tion of the Chebyshev filter presented in [10] for node classification on large-scale graphs. This approach bridges the gap between spectral and spacial approaches, since the convolutions are performed in the spacial domain while being rooted in the spectral domain. The authors show that the convolved node representations computed by their approach can be interpreted as the graph coloring returned by the 1-dimensional WL algorithm [47]. A related node classification approach that is invariant to graph isomorphism is presented in [1]. [12] introduces a graph convolution approach for graph-level classification that generalizes molecular fingerprint design through the use of a differentiable neural network. The approach, however, does not scale to graphs with wide node degree distributions due to the use of node degree-specific trainable weight matrices. A popular deep learning approach for graphs, PATCHY-SAN, is presented in [31]. The authors define of a spatial graph convolution operator that extracts normalized local "patches" (neighborhood representations) of the graph which are then sorted and fed to a 1-dimensional traditional convolution layer to perform graph-level classification. The method, however, requires the definition of a node ordering, as well as running the WL algorithm, in a preprocessing step. The normalization of the extracted parches, on the other hand, implies sorting the nodes again and using the external graph software NAUTY [27]. A related GCN that is invariant to node permutation has been recently presented in [50]. The convolution operator is closely related to the first-order approximation of the Chebyshev filter presented in [21], and the authors introduce a SortPooling operator that sorts the convolved nodes, which are then fed to a 1-dimensional classical convolution layer for graph-level classification. Graph neural networks have been largely applied in chemistry as well. In [13], the authors unify existing graph neural networks for supervised learning on molecules under a common message passing framework. Following this framework, [37] introduces a new graph convolutional architecture that extends the work in [21,12] to link prediction and node classification for large-scale relational multigraphs.

Our approach extends the approximate Chebyshev filter of [21] to graph-level classification by introducing a sum-pooling operator to obtain graph-level representations. Our approach is also related to [50] in that we adopt the same graph convolution operator inspired by the one in [21]. Unlike [50], however, our approach does not require the definition of any node ordering, yet it retains the permutation-invariance property as demonstrated in Section 4, and we only use a simple MLP to perform classification. Our work is also related to [41,1] that use summing-based pooling operators and, hence, are also invariant to node permutation. Both approaches, however, are memory-consuming (in [41], a weight matrix is generated for each edge while [1] requires storing power series of the adjacency matrix) and, consequently, do not scale to very large graphs.

3 Some Graph Concepts

We follow the graph theory framework to formally define the graph-structured data that we handle in this paper and to introduce some graph-related concepts.

A graph G is a pair (V, E) of a set $V = \{v_1, \ldots, v_n\}$ of *vertices* (or *nodes*) v_i, and a set $E \subseteq V \times V$ of *edges* (v_i, v_j). In practice, a graph G is often represented by an *adjacency matrix* $\mathrm{A} \in \mathbb{R}^{n \times n}$ modelling the edges of the graph, where $n = |V|$ is the number of vertices, and such that $a_{ij} = 1$ if there is an edge between nodes v_i and v_j ($(v_i, v_j) \in E$) and $a_{ij} = 0$ otherwise. Edges can be either oriented, and we say that the graph is *directed*, or non-oriented, in which case we say that the graph is *undirected*. Note that the adjacency matrix of undirected graphs is symmetric. In an undirected graph, we say that two vertices v_i and v_j are *neighbors* if there exists and edge $(v_i, v_j) \in E$, and we denote $N(i)$ the *neighborhood* of v_i, i.e. the set of the indices of all neighbors of v_i. The number of neighbors, $|N(i)|$, of a node v_i is called the *degree* of v_i. In directed graphs, similar notions of *indegree* and *outdegree* exist. Finally, edges of the form (v_i, v_i), i.e. edges between a node and itself, are referred to as *self-loops*.

We assume that a graph G is characterized by a *node feature matrix* $\mathrm{X} \in \mathbb{R}^{n \times d}$, with d being the number of node features, in addition to its adjacency matrix A.[4] Each row $\mathrm{x}_i \in \mathbb{R}^d$ of X contains the feature representation of a node v_i, where d is the dimension of the feature space. Since we only consider node features in this paper (as opposed to *edge features* for instance), we will refer to the node feature matrix X simply as the *feature matrix* in the rest of this paper.

We now introduce the notion of *graph isomorphism* in Definition 1.

Definition 1 (Graph Isomorphism). *Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic if there exists a bijection $g : V_1 \to V_2$ such that every edge (u, v) is in E_1 if and only if the edge $(g(u), g(v))$ is in E_2.*

Informally, Definition 1 states that two graphs are isomorphic if there exists a vertex permutation such that when applied to one graph, we recover the vertex and edge sets of the other graph. That is, two graphs are isomorphic if they have the same structure independently of the vertex indexing.

The problem of determining whether two graphs are isomorphic is called the graph isomorphism (GI) problem and is an important one in graph and complexity theory. It is known to be in the class NP and has been largely studied since the 1970's [33]. The notion of graph isomorphism is also directly related to the important notion of invariance to node permutation, as we discuss in the next section.

4 Simple Permutation-Invariant Graph Convolutional Network (SPI-GCN)

SPI-GCN's architecture mainly consists of the following sequential modules:

1) A *graph convolution module* that encodes local graph structure and node features in a substructure feature matrix whose rows represent the nodes of the graph;

[4] We assume without loss of generality that node features are real-valued. Our architecture, however, can handle both real-valued and discrete-valued node features.

2) A *sum-pooling layer* that transforms the substructure feature matrix computed previously into a single feature vector representation of the input graph by summing its rows;
3) A *prediction module* consisting of dense layers that reads the vector representation of the graph and outputs predictions.

Figure 1 summarizes the general architecture of SPI-GCN.

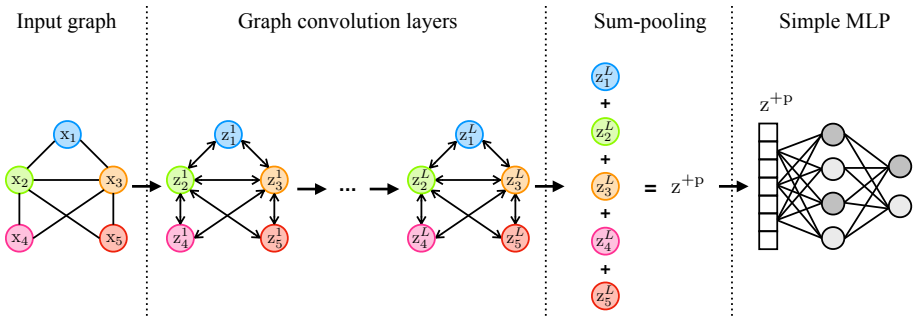

Fig. 1. The general structure of SPI-GCN. The graph convolution layer extracts successive substructure feature matrices from the input graph. The last substructure feature matrix is presented to the sum-pooling layer which sums the feature vectors representing the nodes of the graph, resulting in a one-vector representation for the input graph. This vector representation is then presented to a simple multilayer perceptron (MLP) to perform prediction.

Let G be a graph represented by the adjacency matrix $\mathrm{A} \in \mathbb{R}^{n \times n}$ and the feature matrix $\mathrm{X} \in \mathbb{R}^{n \times d}$, where n and d represent the number of nodes and the dimension of the feature space respectively. Without loss of generality, we consider graphs without self-loops, i.e. the adjacency matrix A has zeros on its diagonal. Additionally, when node features are not available (purely structural graphs), we take $\mathrm{X} = \mathrm{I}_n$, where $\mathrm{I}_n \in \mathbb{R}^{n \times n}$ is the identity matrix.

Notations. We adopt the following notations in the rest of the paper. Matrices are denoted using capital letters while vectors are denoted using small letters. Scalars, on the other hand, are denoted using small italic letters. Let consider a matrix M. m_i denotes its ith row (vector) and m_{ij} denotes the entry at its ith row and jth column. Its inverse matrix is denoted M^{-1}.

4.1 Graph Convolution Module

Given a graph G defined by an adjacency matrix A and a feature matrix X, we define the first convolution layer as follows:

$$\mathrm{Z} = f(\hat{\mathrm{D}}^{-1}\hat{\mathrm{A}}\mathrm{X}\mathrm{W}) \ , \tag{1}$$

where $\hat{\mathrm{A}} = \mathrm{A} + \mathrm{I}_n$ is the adjacency matrix of G with added self-loops, $\hat{\mathrm{D}}$ is the diagonal node degree matrix of $\hat{\mathrm{A}}$,[5] i.e. $\hat{d}_{ii} = \sum_{j=1}^{n} \hat{a}_{ij}$, for $i = 1, \ldots, n$, $\mathrm{W} \in \mathbb{R}^{d \times d'}$ is a trainable weight matrix with d' columns, $f : \mathbb{R}^{n \times d'} \to \mathbb{R}^{n \times d'}$ is a nonlinear activation function, and $\mathrm{Z} \in \mathbb{R}^{n \times d'}$ is the output convolved graph.

The graph convolution defined in (1) extracts local structure and feature information as follows. First, self-loops are added to the graph via $\hat{\mathrm{A}}$ to include each node's features in the convolution. The modified adjacency matrix $\hat{\mathrm{A}}$ is then multiplied by X to produce a matrix $\hat{\mathrm{X}} := \hat{\mathrm{A}}\mathrm{X} \in \mathbb{R}^{n \times d}$ such that

$$\hat{\mathrm{x}}_i = \mathrm{x}_i + \sum_{j \in N(i)} \mathrm{x}_j, \ i = 1, \ldots, n \ . \tag{2}$$

That is, the ith row of $\hat{\mathrm{X}}$ represents the sum of the feature vector of node i and the feature vectors of its neighboring nodes. We then multiply $\hat{\mathrm{X}}$ by $\hat{\mathrm{D}}^{-1}$ to keep the aggregated features on the same scale. The resulting $n \times d$ matrix is then mapped to a new d'-dimensional feature space through multiplication by W. Finally, a nonlinear activation function f is applied element-wise resulting in a $n \times d'$ substructure feature matrix Z that contains the convolution result.

To stack multiple convolution layers, we generalize the propagation rule in (1) as follows:

$$\mathrm{Z}^{l+1} = f^l(\hat{\mathrm{D}}^{-1}\hat{\mathrm{A}}\mathrm{Z}^l\mathrm{W}^l) \ , \tag{3}$$

where $\mathrm{Z}^0 = \mathrm{X}$, $\mathrm{Z}^l \in \mathbb{R}^{n \times d_l}$ is the output of the lth convolution layer with d_l being the number of output features at layer l, $\mathrm{W}^l \in \mathbb{R}^{d_l \times d_{l+1}}$ is a trainable weight matrix that maps the features in Z^l from a d_l-dimensional space to a d_{l+1}-dimensional one, and f^l is the nonlinear activation function applied at level l. Each row of the resulting matrix $\mathrm{Z}^{l+1} \in \mathbb{R}^{n \times d_{l+1}}$ contains a node representation in a new feature space.

Once the input graph has been propagated through all the graph convolution layers, we return the result of the last layer. That is, for a network with L convolution layers, the result of the convolution is the last substructure feature matrix Z^L. Note that (3) accepts graphs with varying node numbers without changing the structure of the convolution layer, i.e. using the same weight matrix W^l. Another important feature of (3) is that the graph convolution computation can be parallelized across the nodes.

Our graph convolution model is inspired by the one proposed in [21] for semi-supervised node classification and where the convolution operator is motivated via spectral graph theory. The idea in [21] is later adopted in [50], where a graph convolution operator very similar to ours is presented. However, while our operator allows the use of different activation functions f^l, as modelled in (3), the one in [50] uses the same nonlinear activation function in all convolution layers. Additionally, [50] use all the substructure feature matrices, Z^l, in the pooling layer whereas we only use the last one, Z^L.

Our graph convolution operator is connected to the 1-dimensional Weisfeiler-Lehman (WL) algorithm [47] as shown in [21,50]. The WL algorithm iteratively

[5] If G is a directed graph, $\hat{\mathrm{D}}$ corresponds to the *outdgree* diagonal matrix of $\hat{\mathrm{A}}$.

computes a vertex coloring for a given graph and is applied in practice to the GI problem. The output of the convolution layer in (3) can be viewed as the vertex coloring computed by the 1-dimensional WL algorithm. This parallel with the WL algorithm allows to define invariant pooling operators such as the SortPooling layer presented in [50] and our sum-pooling layer that we define below.

4.2 Sum-Pooling Layer

The graph convolution module returns a node-level representation $\mathrm{Z}^L \in \mathbb{R}^{n \times d_L}$ of an input graph G, for a module with L convolution layers. The *sum-pooling* layer produces a graph-level representation $\mathrm{z}^{+\mathrm{p}}$ by summing the rows of Z^L according to

$$\mathrm{z}^{+\mathrm{p}} = \sum_{i=1}^{n} \mathrm{z}_i^L \ . \tag{4}$$

The resulting vector $\mathrm{z}^{+\mathrm{p}} \in \mathbb{R}^{d_L}$ contains the final vector representation (or *embedding*) of the input graph G in a d_L-dimensional space. This vector representation is then used for prediction—graph classification in our case.

Using the sum-pooling layer in (4) results in the invariance of our architecture to node permutation as stated in Theorem 1. This invariance property is crucial as it ensures that SPI-GCN will produce the same output for two isomorphic—and hence equivalent—graphs.

Theorem 1. *Let G and G_ς be two arbitrary isomorphic graphs. The sum-pooling layer of SPI-GCN produces the same vector representation for G and G_ς.*

Proof. Let A and A_ς be the adjacency matrices of G and G_ς respectively. A_ς is obtained by interchanging the ith and jth rows and columns of A for every permuted nodes i and j of G. Let Z^l and Z_ς^l be the substructure feature matrices corresponding to G and G_ς respectively and output by the $(l-1)$th graph convolution layer defined in (3). Similarly to A_ς, Z_ς^l is defined by interchanging rows i and j of Z^l for every permuted nodes i and j of G. It is then easy to see that the matrix $\mathrm{A}_\varsigma \mathrm{Z}_\varsigma^l$ corresponds to the matrix $\mathrm{A}\mathrm{Z}^l$ where we interchange rows corresponding to permuted nodes. Note that this result holds when A and A_ς are replaced by the normalized adjacency matrices $\hat{\mathrm{D}}^{-1}\hat{\mathrm{A}}$ and $\hat{\mathrm{D}}_\varsigma^{-1}\hat{\mathrm{A}}_\varsigma$ respectively. Therefore, multiplying $\hat{\mathrm{D}}^{-1}\hat{\mathrm{A}}\mathrm{Z}^l$ and $\hat{\mathrm{D}}_\varsigma^{-1}\hat{\mathrm{A}}_\varsigma\mathrm{Z}_\varsigma^l$ by the same weight matrix W^l, then applying the same nonlinear activation function f^l, will result in two matrices Z^{l+1} and $\mathrm{Z}_\varsigma^{l+1}$ with the same rows ordered differently. Since the sum-pooling layer in (4) simply sums the rows of Z^{l+1} and $\mathrm{Z}_\varsigma^{l+1}$, we end up with the same vector representation for G and G_ς. □

Using a summing-based pooling operator (e.g. sum or average of node features) is a simple idea that has already been implemented in graph neural networks such as [1,41]. The key advantage of summing-based methods is their efficiency and inherent invariance to node permutation. Their main drawback,

on the other hand, is that by summing node features, we lose more refined information on individual nodes and on the global structure of the graph. We show through our work, however, that when combined with the right graph convolution, summing-based architectures are competitive with more complex deep learning graph architectures.

4.3 Prediction Module

The prediction module of SPI-GCN consists in a simple multilayer perceptron (MLP), i.e. fully connected linear layers followed by nonlinear activation functions. It takes as input the graph-level representation $\mathrm{z}^{+\mathrm{p}} \in \mathbb{R}^{d_L}$ returned by the sum-pooling layer and returns either (i) a probability $p \in [0, 1]$ in case of binary classification or (ii) a vector $\mathrm{p} \in \mathbb{R}^C$ of probabilities such that $\sum_{i=1}^{C} p_i = 1$ in case of C-class classification, with $C > 2$.

As a consequence of the architecture presented above, SPI-GCN can be trained in an end-to-end fashion through backpropagation. Moreover, since only one graph is treated in a forward pass, the training complexity of SPI-GCN is linear in the number of graphs.

In the next section, we assess the performance of SPI-GCN and compare it with more complex approaches, including two recent graph deep learning methods and a state-of-the-art graph kernel.

5 Experiments

We evaluate the performance of SPI-GCN on benchmark data sets for graph classification and on an original real-world data set of metal hydrides that we refer to as HYDRIDES. We compare our approach with one state-of-the-art graph kernel and two recent deep learning approaches for graph-structured data. Our PyTorch [32] implementation of SPI-GCN, as well as the data, are available at https://github.com/asmaatamna/SPI-GCN.

5.1 Data Sets

Benchmark data sets We use nine public benchmark data sets to evaluate the accuracy of SPI-GCN. These data sets include five bioinformatics data sets (MUTAG [9], PTC [42], ENZYMES [4], NCI1 [46], and PROTEINS [11]), two social network data sets (IMDB-BINARY and IMDB-MULTI [48]), one image data set where images are represented as region adjacency graphs (COIL-RAG [34]), and one synthetic data set (SYNTHIE [30]).[6] These data sets are available at [19] in a specific text format that we process in order to transform the graphs into a (adjacency matrix, feature matrix) format that can be processed by our neural network. Table 1 summarizes the total number of graphs, the number of classes, the average node number (avg. n), the maximum node number (max. n), and the number of node features d for each data set.

[6] Due to space limitations, we refer the reader to the indicated references for more details on the tested data sets.

HYDRIDES data set HYDRIDES is a real-world data set designed by materials scientists at the ICMPE that contains graph representations of metal hydrides. Metal hydrides play a central role in modern chemistry as they can be used as a storage compound for hydrogen. A very active research field in materials science focuses on synthesizing *stable* metal hydrides suitable for efficient hydrogen storage. A metal hydride is a chemical compound consisting of one or more hydrogen (H) atoms bonded with one or more metallic atoms. Our metal hydrides are *binary*, that is, all metallic atoms are of the same type (e.g. H-Ni compounds, H-Pd compounds, etc). A metal hydride has a *crystal structure* composed of a *unit cell*—a set of atoms arranged in a particular way in the 3-dimensional Euclidean space—that is repeated periodically. Hence, a metal hydride can be fully described by the Euclidean coordinates of the atoms in its unit cell. In crystallography, this geometric information is usually encoded in the so-called POSCAR format that can be used for DFT calculations [24] which determine electronic and stability properties of a given compound, among other properties. DFT calculations, however, are very time consuming, hence the usefulness of machine learning approaches to accelerate decision making. Our HYDRIDES data set contains graphs corresponding to binary compounds that we collect in POSCAR format from the public Pearson's crystal database [44], as well as from [5]. Hydrides are classified into two categories—stable or non-stable—depending on their standard heat of formation. To convert the POSCAR format to our (adjacency matrix, feature matrix) format, we proceed as follows:

- Each unit cell—and hence each metal hydride—is represented by an undirected graph;
- The nodes of the graph correspond to the atoms of the unit cell;
- An edge is created between two nodes if one is the other's nearest neighbor in terms of Euclidean distance;
- Each node is represented by a d-dimensional one-hot vector, where $d = 28$ corresponds to the atom types available in our data set (H combined with 27 metals).

The properties of the HYDRIDES data set are summarized in Table 1.

5.2 Experimental Set-Up

Network architecture The instance of SPI-GCN that we use for experiments has two graph convolution layers of 128 and 32 hidden units respectively, followed by a hyperbolic tangent function (tanh) and a softmax function (per node) respectively. The sum-pooling layer is a classical sum applied row-wise; it is followed by a prediction module consisting of a MLP with one hidden layer of 256 hidden units followed by a batch normalization layer [18] and a rectified linear unit (ReLU) as activation function. We choose this architecture by trial and error, and we keep it unchanged throughout the experiments.

Table 1. Properties of the tested data sets.

Property	Graphs	Classes	Avg. n	Max. n	d
MUTAG	188	2	17.93	28	–
PTC	344	2	14.29	64	–
ENZYMES	600	6	32.63	126	18
NCI1	4110	2	29.87	111	–
PROTEINS	1113	2	39.06	620	1
IMDB-BINARY	1000	2	19.77	136	–
IMDB-MULTI	1500	3	13	89	–
COIL-RAG	3900	100	3.01	11	64
SYNTHIE	400	4	95	100	1
HYDRIDES	1020	2	7.40	70	28

Baselines We compare SPI-GCN with the state-of-the-art Weisfeiler-Lehman subtree kernel (WL) [39], the well-known graph neural network PATCHY-SAN (PSCN) [31], and the more recent deep learning approach Deep Graph Convolutional Neural Network (DGCNN) [50] that uses a very similar convolution module to ours.

Experimental procedure We train SPI-GCN using full batch ADAM optimizer [20], with cross entropy as the loss function to minimize. After trying few combination of values, we set ADAM's hyperparameters as follows. The algorithm is trained for 200 epochs on all benchmark data sets and for 500 epochs on HYDRIDES, and the learning rate is set to 10^{-3}. To estimate the accuracy, we perform 10-fold cross validation using 9 folds for training and one fold for testing each time. We report the average (test) accuracy and the corresponding standard deviation in Table 2. Note that we only use node attributes in our experiments. In particular, SPI-GCN does not exploit node or edge labels of the data sets. When node attributes are not available, we use the identity matrix as the feature matrix ($\mathrm{X} = \mathrm{I}_n$) for each graph.

We follow the same procedure for DGCNN. We use the authors' PyTorch implementation [49] and perform 10-fold cross validation with the recommended values for training epochs, learning rate, and the SortPooling parameter k, for each data set. These values are reported in Table 4.

For PSCN, we report the results from the original paper [31] (for receptive field size $k = 10$) as we could not find an authors' public implementation of the algorithm. The experiments were conducted using a similar procedure as ours.

For WL, we follow [31,48] and set the height parameter h to 2. We choose the regularization parameter C of the SVM from $\{10^{-7}, 10^{-5}, \ldots, 10^{7}\}$ using cross validation as follows: we split the data set into a training set (90% of the graphs) and a test set (remaining 10%), then perform 10-fold cross validation on the training set with LIBSVM [7]. The parameter C with the highest average validation accuracy is then evaluated on the test set. The experiment is repeated 10 times and we report the average test accuracy and the standard deviation. We test the algorithm without using node labels (WL), then with node labels ($\mathrm{WL}_{\mathrm{nl}}$).

We use the authors' MATLAB implementation [38] where we modify the cross validation script to implement the evaluation procedure described previously.[7]

5.3 Results

Table 2 shows the results for our algorithm (SPI-GCN), DGCNN [50], PSCN [31], and WL [39] when node labels are not used. We observe that SPI-GCN is highly competitive with other algorithms despite using the same architecture for all data sets and not tuning the hyperparameters. The only noticeable exceptions are on the NCI1 and IMDB-BINARY data sets, where the best approach (PSCN and WL respectively) is up to 1.19 times better. On the other hand, SPI-GCN performs better than DGCNN and WL on classification tasks with more than 3 classes (ENZYMES, COIL-RAG, SYNTHIE). The difference in accuracy is particularly significant on COIL-RAG (100 classes), where SPI-GCN is around 34.26 times better than DGCNN. This suggests that the features extracted by SPI-GCN are more suitable to characterize the graphs at hand. Results for WL on COIL-RAG and SYNTHIE are not available as we could not find these data sets in the appropriate format for the algorithm online. SPI-GCN also achieves a very reasonable accuracy on the HYDRIDES data set.

We also compare SPI-GCN with WL when node labels are exploited ($\mathrm{WL_{nl}}$). The results, listed in Table 3, show that while the accuracy of WL significantly improves on the PROTEINS and ENZYMES data sets (up to 1.32 times), SPI-GCN remains competitive with $\mathrm{WL_{nl}}$, and even better on the MUTAG data set. Note that no node labels are available for the IMDB-BINARY and IMDB-MULTI data sets, hence the absence of results for $\mathrm{WL_{nl}}$.

We expect the accuracy (respectively variance) of SPI-GCN to improve (respectively decrease) after tuning its hyperparameters to individual data sets. The exploitation of node labels (as additional features) and edge labels (as weights in the adjacency matrix) may also benefit SPI-GCN, especially on data sets where it lags behind other approaches, such as NCI1 and IMDB-BINARY.

6 Conclusion

We were motivated by the development of a principled deep learning approach for graph classification. We proposed an original graph convolutional neural network, SPI-GCN, that is able to process arbitrary graphs directly without any preprocessing, and that is invariant to graph isomorphism thanks to the use of a simple sum-pooling operator. Unlike related deep learning approaches, SPI-GCN has a simple architecture that does not require to sort graphs' vertices and that uses a simple multilayer perceptron to perform classification. Nonetheless, SPI-GCN is competitive with state-of-the-art graph kernels and outperforms similar deep learning approaches on multiclass classification tasks in terms of predictive

[7] The original script returns the average test accuracy of the best C parameters, i.e. parameters with the best validation accuracy on each fold, for one complete run of 10-fold cross validation.

Table 2. Accuracy results for SPI-GCN, two deep learning methods (DGCNN, PSCN), and a graph kernel method (WL without node labels).

Algorithm	SPI-GCN	DGCNN	PSCN	WL
MUTAG	84.40 ± 8.14	86.11 ± 7.14	$\mathbf{88.95 \pm 4.37}$	81.67 ± 15.50
PTC	56.41 ± 5.71	55.00 ± 5.10	$\mathbf{62.29 \pm 5.68}$	56.76 ± 5.89
NCI1	64.11 ± 2.37	72.73 ± 1.56	$\mathbf{76.34 \pm 1.68}$	71.58 ± 2.63
PROTEINS	72.06 ± 3.18	72.79 ± 3.58	$\mathbf{75.00 \pm 2.51}$	68.73 ± 4.24
ENZYMES	$\mathbf{50.17 \pm 5.60}$	47.00 ± 8.36	–	38.67 ± 4.83
IMDB-BINARY	60.40 ± 4.15	68.60 ± 5.66	71.00 ± 2.29	$\mathbf{72.10 \pm 5.30}$
IMDB-MULTI	44.13 ± 4.61	45.20 ± 3.75	45.23 ± 2.84	$\mathbf{51.26 \pm 4.31}$
COIL-RAG	$\mathbf{75.72 \pm 3.65}$	2.21 ± 0.33	–	–
SYNTHIE	$\mathbf{71.00 \pm 6.44}$	54.25 ± 4.34	–	–
HYDRIDES	82.25 ± 3.29	–	–	–

Table 3. Accuracy results for SPI-GCN and the Weisfeiler-Lehman subtree kernel with (WL_{nl}) and without (WL) node labels.

Algorithm	SPI-GCN	WL_{nl}	WL
MUTAG	$\mathbf{84.40 \pm 8.14}$	82.77 ± 8.46	81.67 ± 15.50
PTC	56.41 ± 5.71	$\mathbf{57.05 \pm 7.61}$	56.76 ± 5.89
NCI1	64.11 ± 2.37	79.87 ± 1.77	71.58 ± 2.63
PROTEINS	72.06 ± 3.18	$\mathbf{72.25 \pm 3.22}$	68.73 ± 4.24
ENZYMES	50.17 ± 5.60	$\mathbf{51.16 \pm 5.33}$	38.67 ± 4.83
IMDB-BINARY	60.40 ± 4.15	–	$\mathbf{72.10 \pm 5.30}$
IMDB-MULTI	44.13 ± 4.61	–	$\mathbf{51.26 \pm 4.31}$

Table 4. Hyperparameters used by DGCNN on each data set.

Parameter	Training epochs	Learning rate	SortPooling k
MUTAG	300	10^{-4}	0.6
PTC	200	10^{-4}	0.6
ENZYMES	500	10^{-4}	0.6
NCI1	200	10^{-4}	0.6
PROTEINS	100	10^{-5}	0.6
IMDB-BINARY	300	10^{-4}	0.9
IMDB-MULTI	500	10^{-4}	0.9
COIL-RAG	500	10^{-4}	0.6
SYNTHIE	500	10^{-4}	0.6

accuracy. In the light of these results, we argue that the effectiveness of a graph neural network relies on that of its graph convolution operator more than it does on the use of conventional deep learning components to perform classification.

On a more general level, we are interested in designing *interpretable* approaches for graph classification. Indeed, the majority of machine learning models are "black boxes" [35], i.e. it is difficult for human experts to explain *why* a model returns a particular output given a certain input data. On the other hand, more complex models are not necessarily more accurate [35]. In this context,

building simple architectures is a step towards a better interpretability. Other research avenues include extending our approach to extremely large-scale graph classification problems and using SPI-GCN as a discriminator in a generative adversarial network (GAN) [14] architecture for graphs.

References

1. Atwood, J., Towsley, D.: Diffusion-convolutional neural networks. In: Proceedings of the 30th International Conference on Neural Information Processing Systems. pp. 2001–2009 (2016)
2. Bojchevski, A., Shchur, O., Zügner, D., Günnemann, S.: NetGAN: Generating graphs via random walks. In: Proceedings of the International Conference on Machine Learning. vol. 80, pp. 609–618 (2018)
3. Borgwardt, K.M., Kriegel, H.P.: Shortest-path kernels on graphs. In: Proceedings of the International Conference on Data Mining. pp. 74–81 (2005)
4. Borgwardt, K.M., Ong, C.S., Schönauer, S., Vishwanathan, S.V.N., Smola, A.J., Kriegel, H.P.: Protein function prediction via graph kernels. Bioinformatics **21**(1), 47–56 (2005)
5. Bourgeois, N., Crivello, J.C., Cedenese, P., Joubert, J.M.: Systematic first-principles study of binary metal hydrides. ACS Combinatorial Science **19**(8), 513–523 (2017)
6. Bruna, J., Zaremba, W., Szlam, A., LeCun, Y.: Spectral networks and locally connected networks on graphs. In: International Conference on Learning Representations (2014)
7. Chang, C.C., Lin, C.J.: LIBSVM: A library for support vector machines. ACM Trans. Intell. Syst. Technol. **2**(3), 1–27 (2011)
8. Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., Kuksa, P.: Natural language processing (almost) from scratch. Journal of Machine Learning Research **12**, 2493–2537 (2011)
9. Debnath, A.K., Lopez de Compadre, R.L., Debnath, G., Shusterman, A.J., Hansch, C.: Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity. Journal of Medicinal Chemistry **34**(2), 786–797 (1991)
10. Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering. In: Advances in Neural Information Processing Systems. pp. 3844–3852 (2016)
11. Dobson, P.D., Doig, A.J.: Distinguishing enzyme structures from non-enzymes without alignments. Journal of Molecular Biology **330**(4), 771–783 (2003)
12. Duvenaud, D.K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., Adams, R.P.: Convolutional networks on graphs for learning molecular fingerprints. In: Advances in Neural Information Processing Systems. vol. 28, pp. 2224–2232 (2015)
13. Gilmer, J., Schoenholz, S.S., Riley, P.F., Vinyals, O., Dahl, G.E.: Neural message passing for quantum chemistry. In: Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017. pp. 1263–1272 (2017)
14. Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A.C., Bengio, Y.: Generative adversarial nets. In: Advances in Neural Information Processing Systems. pp. 2672–2680 (2014)

15. Gori, M., Monfardini, G., Scarselli, F.: A new model for learning in graph domains. In: Proceedings of the International Joint Conference on Neural Networks. vol. 2, pp. 729–734 (2005)
16. Haussler, D.: Convolution kernels on discrete structures. Technical Report UCS-CRL-99-10, University of California, Santa Cruz (1999)
17. Henaff, M., Bruna, J., LeCun, Y.: Deep convolutional networks on graph-structured data (2015), arXiv:1506.05163
18. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: Proceedings of the 32Nd International Conference on International Conference on Machine Learning. vol. 37, pp. 448–456 (2015)
19. Kersting, K., Kriege, N.M., Morris, C., Mutzel, P., Neumann, M.: Benchmark data sets for graph kernels (2016), http://graphkernels.cs.tu-dortmund.de
20. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: Proceedings of the 3rd International Conference on Learning Representations (2015)
21. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: International Conference on Learning Representations (2017)
22. Kohn, W.: Electronic structure of matter-wave functions and density functionals. Rev. Mod. Phys. **71**(5), 1253–1266 (1999)
23. Kondor, R., Pan, H.: The multiscale laplacian graph kernel. In: Advances in Neural Information Processing Systems. pp. 2990–2998 (2016)
24. Kresse, G., Joubert, D.: From ultrasoft pseudopotentials to the projector augmented-wave method. Physical Review B **59**(3), 1758–1775 (1999)
25. Kriege, N.M., Johansson, F.D., Morris, C.: A survey on graph kernels (2019), arXiv:1903.11835
26. Kudo, T., Maeda, E., Matsumoto, Y.: An application of boosting to graph classification. In: Advances in Neural Information Processing Systems. vol. 18, pp. 729–736 (2005)
27. Mckay, B.D., Piperno, A.: Practical graph isomorphism, ii. Journal of Symbolic Computation **60**, 94–112 (2014)
28. Min, S., Lee, B., , Yoon, S.: Deep learning in bioinformatics. Briefings in Bioinformatics **18**(5), 851–869 (2017)
29. Morris, C., , Kersting, K., Mutzel, P.: Glocalized weisfeiler-lehman graph kernels: Global-local feature maps of graphs. In: IEEE International Conference on Data Mining. pp. 327–336 (2017)
30. Morris, C., Kriege, N.M., Kersting, K., Mutzel, P.: Faster kernels for graphs with continuous attributes via hashing. In: IEEE 16th International Conference on Data Mining. pp. 1095–1100 (2016)
31. Niepert, M., Ahmed, M., Kutzkov, K.: Learning convolutional neural networks for graphs. In: Proceedings of the 33rd International Conference on International Conference on Machine Learning. vol. 48, pp. 2014–2023. JMLR (2016)
32. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in PyTorch. In: NIPS Autodiff Workshop (2017)
33. Read, R.C., Corneil, D.G.: The graph isomorphism disease. Journal of Graph Theory **1**(4), 339–363 (1977)
34. Riesen, K., Bunke, H.: IAM graph database repository for graph based pattern recognition and machine learning. In: Proceedings of the 2008 Joint IAPR International Workshop on Structural, Syntactic, and Statistical Pattern Recognition. pp. 287–297. Springer Berlin Heidelberg (2008)

35. Rudin, C.: Please stop explaining black box models for high stakes decisions (2018), neurIPS Workshop on Critiquing and Correcting Trends in Machine Learning. arXiv:1811.10154
36. Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The graph neural network model. Transactions on Neural Networks **20**(1), 61–80 (2009)
37. Schlichtkrull, M., Kipf, T.N., Bloem, P., van den Berg, R., Titov, I., Welling, M.: Modeling relational data with graph convolutional networks. In: The Semantic Web (ESWC). pp. 593–607 (2018)
38. Shervashidze, N., Borgwardt, K.M.: Matlab implementation of the weisfeiler-lehman subtree graph kernel, https://www.bsse.ethz.ch/mlcb/research/machine-learning/graph-kernels/weisfeiler-lehman-graph-kernels.html
39. Shervashidze, N., Schweitzer, P., van Leeuwen, E.J., Mehlhorn, K., Borgwardt, K.M.: Weisfeiler-lehman graph kernels. Journal of Machine Learning Research **12**, 2539–2561 (2011)
40. Shervashidze, N., Vishwanathan, S.V.N., Petri, T., Mehlhorn, K., Borgwardt, K.: Efficient graphlet kernels for large graph comparison. In: Proceedings of the International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research, vol. 5, pp. 488–495 (2009)
41. Simonovsky, M., Komodakis, N.: Dynamic edge-conditioned filters in convolutional neural networks on graphs. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2017). pp. 29–38 (2017)
42. Srinivasan, A., Helma, C., King, R.D., Kramer, S.: The predictive toxicology challenge 2000–2001. Bioinformatics **17**(1), 107–108 (2001)
43. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph attention networks. International Conference on Learning Representations (2018)
44. Villars, P., Cenzual, K.: Pearson's crystal data crystal structure database for inorganic compounds. ASM International, http://www.crystalimpact.com/pcd/
45. Vishwanathan, S.V.N., Schraudolph, N.N., Kondor, R., Borgwardt, K.M.: Graph kernels. Journal of Machine Learning Research **11**, 1201–1242 (2010)
46. Wale, N., Watson, I.A., Karypis, G.: Comparison of descriptor spaces for chemical compound retrieval and classification. Knowledge and Information Systems **14**(3), 347–375 (2008)
47. Weisfeiler, B., Lehman, A.A.: A reduction of a graph to a canonical form and an algebra arising during this reduction. Nauchno-Technicheskaya Informatsia **9**(2), 12–16 (1968)
48. Yanardag, P., Vishwanathan, S.: Deep graph kernels. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 1365–1374. ACM (2015)
49. Zhang, M.: Pytorch implementation of Deep Graph Convolutional Neural Network (Dgcnn), https://github.com/muhanzhang/pytorch_DGCNN
50. Zhang, M., Cui, Z., Neumann, M., Chen, Y.: An end-to-end deep learning architecture for graph classification. In: AAAI Conference on Artificial Intelligence. pp. 4438–4445. AAAI Press (2018)