Graph Classification with 2D Convolutional Neural Networks

Antoine J.-P. Tixier
École Polytechnique
antoine.tixier-1@colorado.edu

Giannis Nikolentzos
École Polytechnique and AUEB
nikolentzos@aueb.gr

Polykarpos Meladianos
École Polytechnique and AUEB
pmeladianos@aueb.gr

Michalis Vazirgiannis
École Polytechnique and AUEB
mvazirg@lix.polytechnique.fr

ABSTRACT

Graph learning is currently dominated by graph kernels, which, while powerful, suffer some significant limitations. Convolutional Neural Networks (CNNs) offer a very appealing alternative, but processing graphs with CNNs is not trivial. To address this challenge, many sophisticated extensions of CNNs have recently been introduced. In this paper, we reverse the problem: rather than proposing yet another graph CNN model, *we introduce a novel way to represent graphs as multi-channel image-like structures that allows them to be handled by vanilla 2D CNNs*. Experiments reveal that our method is more accurate than state-of-the-art graph kernels and graph CNNs on 4 out of 6 real-world datasets (with and without continuous node attributes), and close elsewhere. Our approach is also preferable to graph kernels in terms of time complexity. Code and data are publicly available[1].

KEYWORDS

Deep Learning, Graphs Kernels, Neural Networks, Convolution, Social Networks, Bioinformatics.

1 GRAPH CLASSIFICATION

Graphs, or networks, are rich, flexible, and universal structures that can accurately represent the interaction among the components of many natural and human-made complex systems.

A central graph mining task is that of *graph* classification (not to be mistaken with *node* classification). The instances are full graphs and the goal is to predict the category they belong to. The applications of graph classification are numerous and range from determining whether a protein is an enzyme or not in bioinformatics, to categorizing documents in NLP, and social network analysis. Graph classification is the task of interest in this study.

---

[1]link will be provided upon acceptance

2 LIMITATIONS OF GRAPH KERNELS

The state-of-the-art in graph classification is currently dominated by a family of methods referred to as *graph kernels*. Graph kernels compute the similarity between two graphs as the sum of the pairwise similarities between some of their substructures, and then pass the similarity matrix computed on the entire dataset to a kernel-based supervised algorithm such as the Support Vector Machine [7] to learn soft classification rules. Graph kernels mainly vary based on the substructures they use, which include random walks [10], shortest paths [2], and subgraphs [28], to cite only a few. While graph kernels have been very successful, they suffer significant limitations:

L1: High time complexity. This problem is threefold: first, populating the kernel matrix requires computing the similarity between every two graphs in the training set (say of size N), which amounts to $N(N-1)/2$ operations. The cost of training therefore increases much more rapidly than the size of the dataset. Second, computing the similarity between a pair of graphs (i.e., performing a single operation) is itself polynomial in the number of nodes. For instance, the time complexity of the shortest path graph kernel is $O(|V_1|^2|V_2|^2)$ for two graphs (V_1, V_2), where $|V_i|$ is the number of nodes in graph V_i. Processing large graphs can thus become prohibitive, which is a serious limitation as big networks abound in practice. Finally, finding the support vectors is $O(N^2)$ when the C parameter of the SVM is small and $O(N^3)$ when it gets large [4], which can again pose a problem on big datasets.

L2: Disjoint feature and rule learning. With graph kernels, the computation of the similarity matrix and the learning of the classification rules are two independent steps. In other words, the features are fixed and not optimized for the task.

L3: Graph comparison is based on small independent substructures. As a result, graph kernels focus on local properties of graphs, ignoring their global structure [24]. They also underestimate the similarity between graphs and suffer unnecessarily high complexity (due to the explosion of the feature space), as substructures are considered to be orthogonal dimensions [32].

3 PROPOSED METHOD

3.1 Proposed method: overview

We propose a simple approach to turn a graph into a multi-channel image-like structure suitable to be processed by a traditional 2D CNN. The process (summarized in Figure 1) can be broken down into 3 steps:
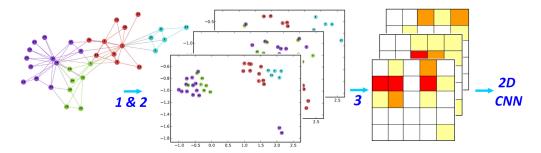
(1) graph node embedding,

Figure 1: Our 3-step approach represents graphs as "images" suitable to be passed to vanilla 2D CNNs (continuous node attribute vectors can be passed as extra channels). Steps 1 & 2: graph node embeddings and compression with PCA. Step 3: computation and stacking of the 2D histograms.

(2) embedding space compression,

(3) repeated extraction of 2D slices from the compressed space and computation of a 2D histogram for each slice.

The "image" representation of the graph is finally given by the stack of its 2D histograms (each histogram making for a channel). Note that the dimensionality of the final representation of a graph does not depend on its number of nodes or edges. Big and small graphs are represented by images of the same size.

Our method addresses the limitations of graph kernels (in **bold**) in the following ways:

L1. By converting all graphs in a given dataset to representations of the same dimensionality, and by using a classical 2D CNN architecture for processing those graph representations, our method offers *constant time* complexity at the instance level, and *linear time* complexity at the dataset level. Moreover, state-of-the-art node embeddings can be obtained for a given graph in *linear time* w.r.t. the number of nodes in the graph, for instance with node2vec [12].

L2. Thanks to the 2D CNN classifier, features are learned directly from the raw data during training to optimize performance on the downstream task.

L3. Our approach capitalizes on state-of-the-art graph node embedding techniques that capture both local and global properties of graphs. In addition, we remove the need for handcrafted features.

3.2 Proposed method: details

How to represent graphs as structures that verify the spatial dependence property?

Convolutional Neural Networks (CNNs) are feedforward neural networks specifically designed to work on regular grids [18]. A regular grid is the d-dimensional Euclidean space discretized by parallelotopes (rectangles for $d = 2$, cuboids for $d = 3$, etc.). Regular grids satisfy the *spatial dependence*[2] property, which is the fundamental premise on which local receptive fields and hierarchical composition of features in CNNs hold.

Traditionally, a graph $G(V, E)$ is encoded as its adjacency matrix A or Laplacian matrix L. A is a square matrix of dimensionality $|V| \times |V|$, symmetric in the case of undirected graphs, whose $(i, j)^{th}$

entry $A_{i,j}$ is equal to the weight of the edge $e_{i,j}$ between nodes v_i and v_j, if such an edge exists, or to 0 otherwise. On the other hand, the Laplacian matrix L is equal to $D - A$, where D is the diagonal degree matrix. One could initially consider passing one of those structures as input to a 2D CNN. However, unlike in images, where close pixels are more strongly correlated than distant pixels, adjacency and Laplacian matrices are not associated with spatial dimensions and the notion of Euclidean distance, and thus do not satisfy the spatial dependence property. As will be detailed next, we capitalize on *graph node embeddings* to address this issue.

Step 1: Graph node embeddings. There is local correlation in the node embedding space. In that space, the Euclidean distance between two points is meaningful: it is inversely proportional to the similarity of the two nodes they represent. For instance, two neighboring points in the embedding space might be associated with two nodes playing the same structural role (e.g., of flow control), belonging to the same community, or sharing some other common property.

Step 2: Alignment and compression with PCA. As state-of-the-art node embedding techniques (such as node2vec) are neural, they are stochastic. Dimensions are thus recycled from run to run, which means that a given dimension will not be associated with the same latent concepts across graphs, or across several runs on the same graph. Therefore, to ensure that the embeddings of all the graphs in the collection are comparable, we apply PCA and retain the first $d \ll D$ principal components (where D is the dimensionality of the original node embedding space). PCA also serves an information maximization (compression) purpose. Compression is desirable as it greatly reduces the shape of the tensors fed to the CNN (for reasons that will become clear in what follows), and thus complexity, at the expense of a negligible loss in information.

Step 3: Computing and stacking 2D histograms. We finally repeatedly extract 2D slices from the d-dimensional PCA node embedding space, and turn those planes into regular grids by discretizing them into a finite, fixed number of equally-sized bins, where the value associated with each bin is the count of the number of nodes falling into that bin. In other words, we represent a graph as a stack of $d/2$ 2D histograms of its (compressed) node embeddings[3]. As illustrated in Figure 2, the first histogram is computed from the coordinates

---

[2] the concept of spatial dependence is well summarized by: "everything is related to everything else, but near things are more related than distant things" [31]. For instance in images, close pixels are more related than distant pixels.

[3] our representation is unrelated to the widespread *color histogram* encoding of images.
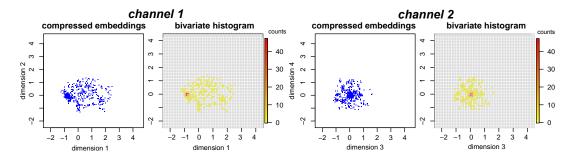
Figure 2: Node embeddings and image representation of graph ID #10001 (577 nodes, 1320 edges) from the REDDIT-12K dataset.

of the nodes in the plane made of the first two principal directions, the second histogram from directions 3 and 4, and so forth. Note that using adjacent and following PCA dimensions is an arbitrary choice. It ensures at least that channels are sorted by decreasing order of informativeness.

Using computer vision vocabulary, bins can be viewed as *pixels*, and the 2D slices of the embedding space as *channels*. However, in our case, instead of having 3 channels (R,G,B) like with color images, we have $d/2$ of them. That is, each pixel (each bin) is associated with a vector of size $d/2$, whose entries are the counts of the nodes falling into that bin in the corresponding 2D slice of the embedding space. Finally, the *resolution* of the image is determined by the number of bins of the histograms, which is constant for a given dataset across all channels.

4 EXPERIMENTAL SETUP

4.1 2D CNN Architecture

We implemented a variant of LeNet-5 [18] with which we reached 99.45% accuracy on the MNIST handwritten digit classification dataset. As illustrated in Figure 3 for an input of shape (5,28,28), this simple architecture deploys four convolutional-pooling layers (each repeated twice) in parallel, with respective region sizes of 3, 4, 5 and 6, followed by two fully-connected layers. Dropout [30] is employed for regularization at every hidden layer. The activations are ReLU functions (in that, our model differs from LeNet-5), except for the ultimate layer, which uses a softmax to output a probability distribution over classes. For the convolution-pooling block, we employ 64 filters at the first level, and as the signal is halved through the (2,2) max pooling layer, the number of filters in the subsequent convolutional layer is increased to 96 to compensate for the loss in resolution.
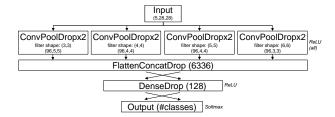

Figure 3: 2D CNN architecture used in our experiments. The number within parentheses refer to the *output* dimensions of the tensors.

4.2 Neural embeddings

To learn node embeddings, we used the high performance C++ implementation[4] of the node2vec algorithm [12]. node2vec applies the very fast Skip-Gram language model [20] to truncated biased random walks performed on the graph. The algorithm scales linearly with the number of nodes in the network.

4.3 Real-world datasets

We conducted experiments on 6 real-world datasets which we describe in what follows, and in Table 1. In all datasets, graphs are unweighted, undirected, with unlabeled nodes, and the task is to predict the class they belong to. Classes are mutually exclusive.

In the first five datasets, we can only learn from the topology of the network, while in the sixth, we also have access to continuous node attributes.

SOCIAL NETWORK DATASETS[5] [32].
Note: in all REDDIT datasets, a graph corresponds to a thread where nodes represent users, and there is an edge between two nodes if one of the two users responded to a comment from the other user.
• **REDDIT-B** graphs are labeled according to whether they were constructed from Q&A or discussion communities,

• **REDDIT-5K** & **REDDIT-12K** feature graphs taken respectively from 5 and 11 forums dedicated to specific topics,

• **COLLAB** graphs are hop-1 neighborhoods of researchers from a scientific collaboration network (two researchers are linked if they co-authored a paper), and are labeled according to the subfield of Physics the corresponding researcher belongs to,

• **IMDB-B** features hop-1 neighborhoods of actors and actresses selected from two movie collaboration networks corresponding to specific genres (action and romance), in which two actors are linked if they starred in the same movie. Graphs are labeled according to the genre they were sampled from.

BIOINFORMATICS DATASET [3, 14] (with continuous node attributes).

• **PROTEINS_full** proteins are represented as graphs where nodes are secondary structure elements (SSEs) and there is an edge between two nodes if they are neighbors in the amino-acid sequence or in 3D space. The goal is to predict whether the protein is an enzyme or not. Furthermore, each node is associated with a **29-dimensional continuous vector** representing chemical measurements.

---

[4]https://github.com/snap-stanford/snap/tree/master/examples/node2vec
[5]http://www.mit.edu/~pinary/kdd/datasets.tar.gz

To incorporate this extra information into our images, we compressed the attribute vectors with PCA, retaining the same number of dimensions as for the node embeddings, and normalized them to have same range as the node embeddings. Finally, each node was represented by a single vector made of its compressed node embedding concatenated with its compressed and normalized continuous attribute vector. That is, we had $d/2$ channels for node embeddings and $d/2$ channels for node attributes, where d is the number of principal components retained in each case.

4.4 Baselines

On the social network datasets (on which there are no continuous node attributes), we re-implemented two state-of-the-art graph kernels, the graphlet kernel and the Weisfeiler-Lehman (WL) kernel.

• the **graphlet kernel** [28] computes the similarity between two graphs as the cosine of their count vectors. These vectors encode how many subgraphs of size up to a certain threshold can be found in each graph (each entry is an occurrence count). We sampled 2000 graphlets of size up to 6 from each graph.

• the **Weisfeiler-Lehman (WL) subtree kernel** [29]. The WL kernel is actually a framework that operates on top of any graph kernel accepting node labels and boosts its performance by using the relabeling procedure of the WL test of isomorphism. More precisely, following the computation of the kernel value between the two graphs, vertex labels are updated based on the labels of their neighbors. This two-step process repeats for a certain number of iterations. The final kernel value is the sum of the values at each iteration. Since our graphs have unlabeled nodes, we use their degrees as labels. Furthermore, we used the WL framework with the subtree graph kernel [10], as it is very efficient with this kernel [29].

We also report for comparison purposes the performance of multiple state-of-the-art baselines (the experimental setting in each case is the same as ours):

• **Deep Graph Kernels** [32]. To be fair, we exclude this baseline from the comparison on the bioinformatics dataset since it doesn't make use of node attributes.

For all datasets:

• **PATCHY-SAN (PSCN $k = 10$)**, which is the best performing graph CNN model presented in [23],

• **Deep Graph CNN (DGCNN)** [33].

On the PROTEINS_full dataset (all baselines below take into account node attribute vectors):

• **HGK-SP** Hash Graph Kernel with shortest-path base kernel [21],

• **HGK-WL** Hash Graph Kernel with Weisfeiler-Lehman subtree base kernel [21],

• **Graph invariant kernels (GIK)** [25] (the best performing variant),

• **GraphHopper** [9] and baselines within (all using continuous node attributes):

• **PROP-diff** propagation kernel with diffusion scheme [22],

• **PROP-WL** propagation kernel with hashing-based label discretization and WL kernel update [22].

4.5 Configuration

Following the conventional experimental set-up, we used 10-fold cross validation and repeated each fold 3 times in all our experiments. For the graphlet and WL kernels, we used a C-SVM classifier[6] [26]. The C parameter of the SVM and the number of iterations in WL were jointly optimized on a 90-10 % partition of the training set of each fold by searching the grid $\{(10^{-4}, 10^4, \text{len} = 10); (2, 7, \text{step} = 1)\}$.

For our 2D CNN, we used Xavier initialization [11], a batch size of 32, and for regularization, a dropout rate of 0.3 and early stopping with a patience of 5 epochs (null delta). The categorical cross-entropy loss was optimized with Adam [15] (default settings). We implemented our model in Keras [6] version 1.2.2[7] with tensorflow [1] backend. The hardware used consisted in an NVidia Titan X Pascal GPU with an 8-thread Intel Xeon 2.40 GHz CPU and 16 GB of RAM, under Ubuntu 16.04.2 LTS 64-bit operating system and Python 2.7. The graph kernel baselines were run on an 8-thread Intel i7 3.4 GHz CPU, with 16 GB of RAM, under Ubuntu 16.06 LTS 64-bit operating system and Python 2.7.

4.6 Resolution, channels, and node2vec parameters

Image resolution. In our initial experiments involving spectral embeddings, the coordinates of any node in any dimension belonged to the $[-1, 1]$ range, due to the eigenvectors being unit-normed. Furthermore, inspired by the MNIST images which are 28×28 in size, and on which we initially tested our 2D CNN architecture, we decided to learn 2D histograms featuring 28 bins in each direction. This gave us a resolution of $28/(1-(-1))$, that is, 14 pixels per unit (or simply 14:1). As it was giving good results, we stuck to similar values in our subsequent experiments making use of neural embeddings. The only other value we experimented with was 9:1.

Number of channels. With the p and q parameters of node2vec held constant and equal to 1, we conducted a search on the coarse grid $\{(14,9);(2,5)\}$ to get more insights about the impact of resolution and number of channels (respectively). When using 5 channels, the graphs with less than 10 nodes were removed, because for these graphs, we cannot get a 10-dimensional node embedding space (we cannot have more dimensions than data points). However, this represented only a couple of graphs overall.
On the PROTEINS_full dataset, we only experimented with 2 node embeddings channels and 2 node attributes channels.

Image size. On a given dataset, image size is calculated as the range $|max(\text{coordinates})-min(\text{coordinates})|\times\text{resolution}$, where coordinates are the flattened node embeddings. For instance, on COLLAB with a resolution of 9:1, image size is equal to 37×37, since $|2.78 - (-1.33)| \times 9 \approx 37$.

p,q,c, and d_{n2v} node2vec parameters. With the best resolution and number of channels, we then tuned the return and in-out parameters p and q of node2vec. Those parameters respectively bias the random walks towards exploring larger areas of the graph or staying in local neighborhoods, allowing the embeddings to encode a similarity that interpolates between structural equivalence (two nodes acting as, e.g., flow controllers, are close to each other) and

---

	IMDB-B	COLLAB	REDDIT-B	REDDIT-5K	REDDIT-12K	PROTEINS_full
Max # vertices	136	492	3782	3648	3782	620
Min # vertices	12	32	6	22	2	4
Average # vertices	19.77	74.49	429.61	508.50	391.40	39.05
Max # edges	1249	40120	4071	4783	5171	1049
Min # edges	26	60	4	21	1	5
Average # edges	96.53	2457.78	497.75	594.87	456.89	72.82
# graphs	1000	5000	2000	4999	11929	1113
Average diameter	1.861	1.864	9.72	11.96	10.91	11.57
Average density (%)	52.06	50.92	2.18	0.90	1.79	21.21
# classes	2	3	2	5	11	2
Max class imbalance	1:1	1:3.4	1:1	1:1	1:5	1:1.5

Table 1: Statistics of the social network datasets (first 5 columns) and the bioinformatics dataset used in our experiments.

homophily (two nodes belonging to the same community are close to each other). Following the `node2vec` paper, we tried 5 combinations of values for (p, q): $\{(1, 1); (0.25, 4); (4, 0.25); (0.5, 2); (2, 0.5)\}$. Note that $p = q = 1$ is equivalent to `DeepWalk` [27].

Since the graphs in the COLLAB and the IMDB-B datasets are very dense (> 50%, average diameter of 1.86), we also tried to set the context size c to smaller values of 1 and 2 (the default value is $c = 10$). The context size is used when generating the training examples (`context,target`) to pass to the skip-gram model: in a given random walk, it determines how many nodes *before* and *after* the target node should be considered part of the context.

Finally, since the graphs in COLLAB, IMDB-B, and PROTEINS_full are small (20, 74, and 39 nodes on average), we experimented with lower values of d_{n2v}, namely 12 and 4. d_{n2v} is the dimension of the node embeddings learned by `node2vec` (default value is 128).

The final values of p, q, c, and d_{n2v} for each dataset are summarized in Table 2.

	REDDIT-B	REDDIT-5K	REDDIT-12K	COLLAB	IMDB-B	PROTEINS_full
Res.	9:1	9:1	9:1	9:1	14:1	9:1
#Chann.	5	2	5	5	5	2/2*
p,q	2,0.5	4,0.25	1,1	0.5,2	1,1	0.5,2
c,d_{n2v}	-	-	-	2,12	-	-,4

Table 2: Final resolution, number of channels, and p, q, c, and d_{n2v} `node2vec` parameters for each dataset. ★number of channels for node embeddings and continuous node attributes. - means default value(s).

5 RESULTS

The classification accuracy of our approach in comparison to the baselines is reported in Table 3 for the social network datasets and Table 4 for the bioinformatics dataset.

Our approach shows (statistically) significantly better than all baselines on the REDDIT-12K and REDDIT-B datasets, with large improvements of **6.81** and **2.82** in accuracy over the best performing competitor, respectively.

We also reach best performance on the REDDIT-5K and PRO-TEINS_full datasets, with respective improvements in accuracy of **1.34** and **0.52** over the best performing baselines. In particular, the fact that we reach best performance on PROTEINS_full shows that

our approach is flexible enough to leverage not only the topology of the network, but also continuous node attributes, in a very simple and unified way (we simply concatenate node embeddings with node attribute vectors). Note that when not using node attributes (only the first 2 node embeddings channels), the performance of our model decreases from 77.12 to 73.43 on PROTEINS_full (3.69 decrease), which proves that the skill of our model comes from leveraging both node embeddings and attributes.

Finally, on the IMDB-B dataset, we get third place, very close ($\leq$ 1.2) to the top performers (no statistically significant difference). The only dataset on which a baseline proved significantly better than our approach is actually COLLAB (WL graph kernel). On this dataset though, the WL kernel beats *all* models by a wide margin, and we are relatively close to the other Deep Learning approaches ($\leq$ 2.43).

5.1 Runtimes

Even if not directly comparable, we report in Table 5 kernel matrix computation time for the two graph kernel baselines, along with the time required by our 2D CNN model to perform one pass over the entire training set, i.e., the time per epoch. With respects to time complexity, our method is superior to graph kernels on several counts: first, unlike graph kernels, the time required by the 2D CNN to process one training example is constant (all images for a given dataset have the same size), while computing the kernel value for a pair of graphs depends on their size (polynomial in the number of nodes). It is true that a prerequisite for our approach is an embedding for all the graphs in the dataset, but `node2vec` scales linearly with the number of nodes in the graph. Therefore, on big graphs, our method is still usable, while graph kernels may not be. Also, `node2vec` is easily parallelizable over the collection, so one can take advantage of multi-core CPUs to considerably speed up the process. Second, with a 2D CNN, the time necessary to go through the entire training set only increases linearly with the size of the set, while populating the kernel matrix is quadratic, and finding the support vectors is then again at least quadratic. This means that on large datasets, our approach is also preferable to graph kernels. Examples of 2D CNN architectures much more complex than ours applied to millions of images in reasonable time abound in the recent computer vision literature. Processing such big

Table 3: 10-fold CV average test set classification accuracy of our proposed method compared to state-of-the-art graph kernels and graph CNNs, on the social network datasets. ± is standard deviation. Best performance per column in bold. ⋆ indicates stat. sign. at the $p < 0.05$ level (our 2D CNN vs. WL) using the Mann-Whitney U test (https://docs.scipy.org/doc/scipy-0.19.0/reference/generated/scipy.stats.mannwhitneyu.html).

Dataset / Method	REDDIT-B (size=2,000;nclasses=2)	REDDIT-5K (4,999;5)	REDDIT-12K (11,929;11)	COLLAB (5,000;3)	IMDB-B (1,000;2)
Graphlet Shervashidze2009	77.26 (± 2.34)	39.75 (± 1.36)	25.98 (± 1.29)	73.42 (± 2.43)	65.40 (± 5.95)
WL Shervashidze2011	78.52 (± 2.01)	50.77 (± 2.02)	34.57 (± 1.32)	77.82⋆ (± 1.45)	**71.60** (± 5.16)
Deep GK Yanardag2015	78.04 (± 0.39)	41.27 (± 0.18)	32.22 (± 0.10)	73.09 (± 0.25)	66.96 (± 0.56)
PSCN $k = 10$ Niepert2016	86.30 (± 1.58)	49.10 (± 0.70)	41.32 (± 0.42)	72.60 (± 2.15)	71.00 (± 2.29)
DGCNN Zhang2018	-	-	-	73.76 (± 0.49)	70.03 (± 0.86)
2D CNN (our method)	**89.12⋆** (± 1.70)	**52.11** (± 2.24)	**48.13⋆** (± 1.47)	71.33 (± 1.96)	70.40 (± 3.85)

	2D CNN our method	DGCNN Zhang18	PSCN $k = 10$ Niepert16	HGK-SP Morris16	HGK-WL Morris16	GIK Orsini15	GraphHopper Feragen13	PROP-diff Neumann12	PROP-WL Neumann12
Acc.	**77.12**	75.54	75.00	75.14	74.88	76.6	74.1	73.3	73.1
Std. dev.	2.79	0.94	2.51	0.47	0.64	0.6	0.5	0.4	0.8

Table 4: 10-fold CV average test set classification accuracy of our proposed method compared to state-of-the-art graph kernels and graph CNNs on the bioinformatics dataset (PROTEINS_full).

	REDDIT-B	REDDIT-5K	REDDIT-12K	COLLAB	IMDB-B	PROTEINS_full
Size, average (# nodes, # edges)	2000, (430,498)	4999, (509,595)	11929, (391,457)	5000, (74,2458)	1000, (20,97)	1113, (39,73)
Input shapes (for our approach)	(5,62,62)	(2,65,65)	(5,73,73)	(5,36,36)	(5,37,37)	(4,70,70) (2,70,70)⋆
Graphlet Shervashidze2009	551	5046	12208	3238	275	-
WL Shervashidze2011	645	5087	20392	1579	23	-
2D CNN (our approach)	6	16	52	5	1	1

Table 5: Runtimes in seconds, rounded to the nearest integer. For the graph kernel baselines, time necessary to populate the Kernel matrix (8-thread 3.4GHz CPU). For our model, time per epoch (Titan X Pascal GPU). ⋆with and without using node attributes

datasets with graph kernels would simply be intractable. In addition, the performance of Deep Learning models tends to significantly improve in the presence of large quantities of training data.

6 RELATED WORK

Motivated by the outstanding performance recently reached by Convolutional Neural Networks (CNNs) in computer vision, e.g. [17?], many research efforts have been devoted to generalizing CNNs to graphs. Indeed, CNNs offer a very appealing alternative to kernel-based methods. The parsimony achieved through weight sharing makes them very efficient, their time complexity is constant for each training example and linear with respect to the size of the dataset, and the extra expressiveness they bring might translate to significant accuracy gains.

However, since convolution and pooling are natively defined for regular, low-dimensional grids such as images (2D Euclidean space discretized by rectangles), generalizing CNNs to graphs, which are irregular, non-Euclidean objects, is far from trivial. Possible solutions that can be found in the literature fall into two broad categories: *spatial* and *spectral* techniques [5]. Spectral approaches [8, 16] invoke the convolution theorem from signal processing theory to perform graph convolutions as pointwise multiplications in the Fourier domain of the graph. The basis used to send the graph

to the Fourier domain is given by the SVD decomposition of the Laplacian matrix of the graph, whose eigenvalues can be viewed as "frequencies". By contrast, spatial methods [23, 33?] operate directly on the graph structure. For instance, in [23], the algorithm first determines the sequence of nodes for which neighborhood graphs (of equal size) are created. To serve as receptive fields, the neighborhood graphs are then normalized, i.e., mapped to a vector space with a linear order, in which nodes with similar structural roles in the neighborhood graphs are close to each other. Normalization is the central step, and is performed via a labeling procedure. A 1D CNN architecture is finally applied to the receptive fields.

6.1 Departure from previous work

While the aforementioned sophisticated frameworks have made great strides, we showed in this paper that graphs can also be processed by vanilla 2D CNN architectures. This is made possible by the novel graph representation we introduce, which encodes graphs as stacks of 2D histograms of their node embeddings (and continuous attribute vectors, if available). Compared to the more complex approaches that involve different architectural and/or operational modifications, the main advantage of our method is its simplicity. Crucially, we show that this simplicity can be obtained *without*

giving up accuracy: we indeed outperform graph CNN baselines by a wide margin on some datasets, and are very close elsewhere.

6.2 Discussion

Replacing the raw counts by the empirical joint probability density function, either by normalizing the histograms, or with a Kernel Density Estimate, significantly deteriorated performance. This suggests that keeping the absolute values of the counts is important, which makes sense, because some categories might be associated with larger or smaller graphs, on average. Therefore, preventing the model from using size information is likely to decrease accuracy. We also observed that increasing the number of channels to more than 5 does not yield better results (which makes sense, as channels contain less and less information), but that reducing this number improves performance in some cases, probably because it plays a regularization role.

The main contribution of our study is a novel method for *representing* graphs as multi-channel image-like structures from their node embeddings, that allows them to be processed by 2D CNNs. How the embeddings are computed, and which 2D CNN architecture is used, does not matter. We hold this flexibility to be a major strength. First, the *embedding-agnostic* nature of our method means that it can be seamlessly extended to directed, weighted, or labeled graphs with continuous or categorical node/edge attributes, simply by using an embedding algorithm that accepts such graphs, e.g., [19]. The independence of our approach with respect to the image classification model used is another advantage. Here, we employed a vanilla 2D CNN architecture as it was offering an excellent trade-off between accuracy and simplicity, but more recent models, such as the one of [13], may yield even better results. Above all, performance should improve as graph node embedding algorithms and CNN architectures for images improve in the future.

Limitations. Even though results are very good out-of-the-box in most cases, finding an embedding algorithm that works well, or the right combination of parameters for a given dataset, can require some efforts. For instance, on COLLAB and IMDB-B (the only two datasets on which we do not reach best performance), we hypothesize that our results are inferior to that observed elsewhere because the default parameter values of node2vec may not be well-suited to very dense graphs such as the ones found in COLLAB and IMDB-B (diameter< 2, density > 50). Optimizing the node2vec parameters on these datasets probably requires more than a coarse grid search.

7 CONCLUSION

The main contribution of this paper is to show that CNN architectures designed for images can be used for graph processing in a completely off-the-shelf manner, simply by representing graphs as stacks of two-dimensional histograms of their node embeddings. Our approach is flexible and continuous node attributes can be taken into account by passing them as additional channels. Despite the simplicity of our approach, we reach better results than state-of-the-art graph kernels and graph CNN models on 4 real-world datasets out of 6. Furthermore, these good results were obtained with limited parameter tuning and by using a basic 2D CNN model. From a time complexity perspective, our approach is preferable to

graph kernels too, allowing to process bigger datasets featuring larger graphs.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467* (2016).
[2] K. M. Borgwardt and H. Kriegel. 2005. Shortest-path kernels on graphs. In *Proceedings of the 5th International Conference on Data Mining*. 74–81.
[3] Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. 2005. Protein function prediction via graph kernels. *Bioinformatics* 21, suppl 1 (2005), i47–i56.
[4] Léon Bottou and Chih-Jen Lin. 2007. Support vector machine solvers. *Large scale kernel machines* 3, 1 (2007), 301–320.
[5] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2013. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203* (2013).
[6] François Chollet et al. 2015. Keras. https://github.com/fchollet/keras. (2015).
[7] Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Machine learning* 20, 3 (1995), 273–297.
[8] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*. 3837–3845.
[9] Aasa Feragen, Niklas Kasenburg, Jens Petersen, Marleen de Bruijne, and Karsten Borgwardt. 2013. Scalable kernels for graphs with continuous attributes. In *Advances in Neural Information Processing Systems*. 216–224.
[10] Thomas Gärtner, Peter Flach, and Stefan Wrobel. 2003. On graph kernels: Hardness results and efficient alternatives. In *Learning Theory and Kernel Machines*. Springer, 129–143.
[11] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks.. In *Aistats*, Vol. 9. 249–256.
[12] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 855–864.
[13] Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. 2016. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993* (2016).
[14] Kristian Kersting, Nils M. Kriege, Christopher Morris, Petra Mutzel, and Marion Neumann. 2016. Benchmark Data Sets for Graph Kernels. (2016). http://graphkernels.cs.tu-dortmund.de
[15] Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
[16] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).
[17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.
[18] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.
[19] Lizi Liao, Xiangnan He, Hanwang Zhang, and Tat-Seng Chua. 2017. Attributed Social Network Embedding. *arXiv preprint arXiv:1705.04969* (2017).
[20] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).
[21] Christopher Morris, Nils M Kriege, Kristian Kersting, and Petra Mutzel. 2016. Faster kernels for graphs with continuous attributes via hashing. In *Data Mining (ICDM), 2016 IEEE 16th International Conference on*. IEEE, 1095–1100.
[22] Marion Neumann, Novi Patricia, Roman Garnett, and Kristian Kersting. 2012. Efficient graph kernels by randomization. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 378–393.
[23] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. 2016. Learning convolutional neural networks for graphs. In *Proceedings of the 33rd annual international conference on machine learning. ACM*.
[24] Giannis Nikolentzos, Polykarpos Meladianos, and Michalis Vazirgiannis. 2017. Matching Node Embeddings for Graph Similarity.. In *AAAI*. 2429–2435.
[25] Francesco Orsini, Paolo Frasconi, and Luc De Raedt. 2015. Graph invariant kernels. In *Proceedings of the Twenty-fourth International Joint Conference on Artificial Intelligence*. 3756–3762.
[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.
[27] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international*

conference on Knowledge discovery and data mining. ACM, 701–710.

[28] N. Shervashidze, T. Petri, K. Mehlhorn, K. M. Borgwardt, and S.V.N. Vishwanathan. 2009. Efficient Graphlet Kernels for Large Graph Comparison. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*. 488–495.

[29] N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn, and K. M. Borgwardt. 2011. Weisfeiler-Lehman Graph Kernels. *The Journal of Machine Learning Research* 12 (2011), 2539–2561.

[30] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* 15, 1 (2014), 1929–1958.

[31] Waldo R Tobler. 1970. A computer movie simulating urban growth in the Detroit region. *Economic geography* 46, sup1 (1970), 234–240.

[32] Pinar Yanardag and SVN Vishwanathan. 2015. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1365–1374.

[33] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. 2018. An End-to-End Deep Learning Architecture for Graph Classification. (2018).