24th European Conference on Artificial Intelligence - ECAI 2020
Santiago de Compostela, Spain

Learning Kernel-based Embeddings in Graph Neural Networks

Nicolò Navarin[1] and **Dinh Van Tran**[2] and **Alessandro Sperduti**[3]

Abstract. We investigate whether Graph Convolutional Neural Networks (GCNNs) may benefit from incorporating information conveyed by a state-of-the-art graph kernel in the learning process. We propose a GCNN architecture and a training procedure based on multi-task learning, where we provide supervision not only from the graph labels, but also from the kernel to each layer of the network, achieving state-of-the-art performances on many real-world datasets. We conduct an ablation study to analyze the impact on the predictive performances of each part of our proposal, including a simplified version of our multi-task learning formulation that can, in principle, be applied with a broad family of graph embeddings. Finally, we study how to improve the performance of a model considering graphs coming from related datasets into the training procedure in a semi-supervised learning fashion.

1 INTRODUCTION

State-of-the-art machine learning techniques for classification and regression on graphs are at the moment kernel machines equipped with specifically designed kernels for graphs (e.g, [31, 36]). Although there are examples of kernels for structures that can be designed on the basis of a training set [35, 3], most of the more efficient and effective graph kernels are based on predefined structural features, i.e, features definition is not part of the learning process. In vectorial, as well as sequential spaces, deep learning has proven that it is actually possible to learn very effective representations, so it is not a surprise that recently many researchers have decided to attack structured domains using (deep) neural networks (NNs), proposing several architectures (see for example [11]), with the aim of getting better performances with respect to graph kernels (which do not learn graph representations). Graph Neural Networks (GNNs), however, not only have to face the well known graph isomorphism problem, i.e. produce an internal representation of a graph that is invariant with respect to its input representation, but unlike graph kernels have also to face the problem of how to explicitly represent in a fixed-size representation graphs of different sizes, i.e. with a variable number of vertices. This last issue is where most of the network architectures proposed up to now try to differentiate each other in order to get a more expressive merging operation (readout) over vertices, so to get a more effective graph representation. Actually, we believe this could be the main bottleneck of the current architectures, which in a way or the other leads to a loss of structural information which inevitably leads to a loss of performance with respect to graph kernels. This is especially true for state-of-the-art graph convolutional networks (e.g. [40]), where the first stages of the architecture (just before the readout) compute a soft version of the Weisfeiler-Lehman (WL) Subtree Kernel [30], i.e. one of the most efficient and effective graph kernels. So a legitimate question is why such a soft version of WL is not always working, i.e. sometimes it shows lower predictive performance compared to WL kernel.

We start from two observations: (i) readout functions make difficult to propagate informative gradients to the convolution layers; (ii) graph kernels perform well, so they can be used to improve the training process in an unsupervised way. Based on these observations, in this paper we propose a novel GCNN architecture, dubbed Funnel GCNN (FGCNN), that tries to remove obstacles to gradient flow in many ways: *i)* using a simple readout and LeakyReLU activation functions; *ii)* forcing the network to reconstruct the corresponding explicit feature space representation of the WL kernel after each convolution layer; *iii)* using a measure of the WL kernel complexity to decide the number of filters (neurons) to use at each convolution layer. While *i)* is a trivial step, in order to perform *ii)* we exploit multi-task learning (MTL) [5] to force the network to learn, for each convolution layer, an approximation of the explicit feature space representation of the WL kernel, hinging on the availability of an explicit WL embedding for each graph. This is possible since WL embeddings are obtained by summing up the features obtained by different WL iterations, that correspond to different depths of the extracted features. This property give us the possibility to split the features generated by the WL kernel according to their complexity (i.e. the WL iteration) and define an output target for each corresponding convolution layer. Finally, the number of filters (neurons) at each convolution layer follows a formal measure of the WL kernel complexity (*iii)*), thus increasing with the depth of the convolution, and giving rise to the funnel shape. We also provide a simple bound on the disagreement in classification between the hypothesis found by an SVM using the WL kernel and the representation learnt by our network. Thanks to this bound, we can guarantee that the performances of the GNN trained with our proposed approach will be (at least) close to the ones of WL kernel. We perform an extensive ablation study in which we analyze the effects on the predictive performance of each main component of our proposed approach.

Finally, noticing that learning the WL embeddings is unsupervised, we explore the effect of adding to the training set unlabelled data, in a semi-supervised fashion. The expected result is an improvement in performances. Preliminary experimental results seem to confirm this, opening the door to an interesting future line of investigation.
In summary, this paper delivers the following original contributions:

- the definition of a general framework, based on multi-task training,

[1] Department of Mathematics, University of Padua, Italy, email: nnavarin@math.unipd.it
[2] School of Computer Science, University of Freiburg, Germany, email: dinh@informatik.uni-freiburg.de
[3] Department of Mathematics, University of Padua, Italy, email: sperduti@math.unipd.it

to embed information conveyed by graph kernels in the hidden representation learned by GNNs;
- a bound on the performance difference between an SVM trained with a graph kernel k and a GNN trained using our proposed framework;
- a novel GNN architecture tailored for the proposed training framework;
- a novel approach for semi-supervised learning for graphs.

2 BACKGROUND

In this section, we review all the basic components of our proposed approach.

Definitions and notations We denote matrices, vectors and variables with bold uppercase, uppercase, and lowercase letters, respectively. Given a matrix $\mathbf{M}$, M_i denotes the i-th row of the matrix, and m_{ij} is the element in i-th row and j-th column. Given the vector V, v_i refers its i-th element.

Let us consider a graph $g = (V^g, E^g, \mathbf{X}^g)$, where $V^g = \{v_1, \ldots, v_n\}$ is the set of vertices (nodes), $E_g \subseteq V^g \times V^g$ is the set of edges, and $\mathbf{X}^g \in \mathbb{R}^{n \times d}$ is a node label matrix, where each row is the label (a vector of size d) associated with each vertex $v_i \in V^g$, i.e. $X^g_i = [x_{i,1}, \ldots, x_{i,d}]$. Here we do not consider edge labels. When the reference to the graph g is clear from the context, for the sake of notation, we discard the superscript referring to the specific graph. We define the adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ as $a_{ij} = 1 \iff (i,j) \in E$, 0 otherwise.

Here, we focus on the problem of graph classification. Given a dataset composed of m pairs $\{(g^{(i)}, y^{(i)}) | 1 \leq i \leq m\}$, the task is then, given an unseen graph g, to predict its real target y. We will consider, for this learning task, graph neural networks and graph kernels, that are discussed in the next sections.

Graph kernels A kernel on $\mathcal{X}$, the input space, is a symmetric positive semi-definite function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ computing a score (*similarity*) between pairs of instances. Kernel functions compute the dot product between two entities in a *Reproducing Kernel Hilbert Space* (RKHS), i.e.: $k(x, y) = \langle \phi(x), \phi(y) \rangle$ where $\phi : \mathcal{X} \to \mathcal{H}$ is a function mapping instances from $\mathcal{X}$ to the RKHS (or feature space) $\mathcal{H}$. Different kernels define different feature spaces (see supplementary material for additional graph kernels).

The approaches proposed in Section 4.2 can, in principle, work for any graph kernel. In this paper, we decide to focus on the WL kernel, that counts the number of identical subtree patterns obtained by breadth-first visits where each node can appear multiple times [30]. The kernel depends on an hyper-parameter h, that is the a-priori selected number of WL iterations, corresponding to the maximum depth of the considered patterns. WL, as well as other kernels [7, 8], allows to explicitly store the $\phi(g)$ representation of a graph g as a sparse vector Φ_g. Moreover, it is also possible to compress Φ_g using a hash function $h : \mathbb{N} \to \{1, ..., b\}$ (or a random projection), obtaining a compact dense representation with a small impact on the predictive performance [21]. We will exploit this property in Section 4.2.1.

Neural networks for graphs The core machine learning models that we are going to adopt in this paper are neural networks for graphs. Our proposed training method can be applied, in principle, with all the models presented in this section. The first definition of neural network for graphs has been proposed in [33], and more recent models have been proposed in [17]. The latter work is based on an idea that has been re-branded later as *graph convolution* or *neural message passing*. The idea is to define the neural architecture following the topology of the graph. Then a transformation is performed from the neurons corresponding to a vertex and its neighborhood to a new hidden representation, that is associated to the same vertex (possibly in another layer of the network). This transformation (graph convolution) depends on some parameters, that are shared among all the vertices. After a certain number of transformations, a *readout* layer merges all graph vertex representations into a fixed-size vector representing the whole graph, from which a fully connected layer can be attached to compute the output. In the following, we review these two basic components of graph neural networks.

Graph convolutions In the following, for the sake of simplicity, we ignore the bias terms. In [29], a recurrent graph neural network is defined as a contraction mapping. In [16], this work has been extended, removing the constraint for the recurrent system to be a contraction mapping, and replacing the recurrent units with GRUs. In [17], the first model resembling a convolutional network for graphs is proposed. This model catches adaptive contextual transductions, learning the mapping from graphs. In [13] a widely adopted formulation of graph convolution is derived. Let us consider $\mathbf{H}^0 = \mathbf{X}$. Motivated by a first-order approximation of localized spectral filters on graphs, the graph convolutional filter looks like: $\mathbf{H}^{i+1} = f(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^i\mathbf{W}^i)$, where $i = 0, \ldots, l-1$ (and l is the number of layers), $\mathbf{W}^i$ is the convolution weight matrix to learn, $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$, $\tilde{d}_{ii} = \sum_j \tilde{a}_{i,j}$, and f is any activation function applied element-wise. In [40] a similar convolution operator derived from the random walk normalized Laplacian, instead of the symmetric normalized graph Laplacian, is defined , while in [18] another variation inspired by the WL isomorphism test is presented. Recently, the graph convolutional filter has been extended with an hyper-parameter controlling the size of the convolution [34]. [23] follows a more straightforward approach to define convolutions on graphs, that is conceptually closer to convolutions defined over images. This approach requires the vertices of each input graph to be in a canonical order, that is as complex as the graph isomorphism problem (no polynomial-time algorithm is known). [2] defines a different graph convolution (i.e. diffusion-convolution) that incorporates in the definition of graph convolution the diffusion operator, i.e. the multiplication of the input representation with a power series of the degree-normalized transition matrix.

Readout functions After stacking a number of graph convolution layers (or graph recurrent layers), we need a mechanism to predict the target for the whole graph, starting from the encoding of its vertices. This mechanism should be applicable to graphs with a variable number of vertices. Various approaches have been proposed in literature.The simplest formulations are linear, namely the average and the sum of vertex representations, possibly followed by dense layers. These readouts are used, for instance, in [17, 10, 2]. In [40], the *SortPooling* readout layer is proposed. The underpinning idea is to select (or *pool*) a fixed number of vertex embeddings, obtaining a fixed-size representation given by the concatenation of them. In [10], it is proposed to use a set2set model which is a simplified Neural Turing Machine for handling sets as inputs. The model is capable of mapping sets to other sets in output, thus it is more powerful than what is required for classification or regression tasks. This complexity makes this readout hard to train, introducing unneeded complexity in the model. In [20], a universal readout based on the sum aggregator is presented.

Pooling layers Recently, local graph pooling operators have been presented, e.g. [4, 39], to reduce the computational complexity of GCNNs.

Mainstream GNN architectures Among the different GNNs proposed in literature, we will consider as reference DGCNN [40], that is one of the best performing GNN architectures to date. The network

consists of three graph convolution layers followed by a concatenation layer that merges the representations at the different levels of graph convolution. The *readout* is composed of a SortPooling layer, followed by two 1D convolutional layers and one dense layer. The activation function for the graph convolutions is the *hyperbolic tangent*, while the 1D convolutions and the dense layer use *rectified linear units* (ReLU).

3 RELATED WORKS

[15] attempts to integrate in the neural network architecture the knowledge acquired from the *design* of kernels. Our proposed FGCNN is also inspired by graph kernels, since its key architectural feature is to have the number of neurons for each layer to increase with depth, mimicking the behavior of graph kernels. However, in this paper we go beyond the simple design of neural architectures, providing a multi-task training approach that additionally provides guarantees on the quality of the learnt hypothesis. In [25], traditional CNNs are applied on top of a graph embedding computed by graph kernels. Differently from this approach, in this paper we deal with neural networks that are able to directly process graphs as their input. We use the kernel in our training approaches for driving the parameter learning procedure. The most related work to the present paper is the workshop paper [19], where the aim is to incorporate information from a graph kernel in the *learning* process of a GCNN by defining a siamese network that, given a graph kernel and a pair of graphs in input, computes an approximation of the kernel value via a dot product unit in output. Learning this kernel approximation constitutes a pre-training phase that is then followed by a standard supervised learning on a single branch of the network equipped with a dense layer for computing the classification in output. Unfortunately, while this approach returns improved performances, it does not scale well with the size of the dataset, since the siamese network takes as input pairs of graphs, thus requiring a quadratic complexity for the pre-training phase.

4 PROPOSED METHOD

While theoretically a GCNN should be able to learn features that are comparable to the ones of WL graph kernel, we will show in Section 5 that this does not always happen in practice. Specifically, on some datasets the WL kernel performs better than state-of-the-art GCNNs. To tackle this problem, we propose a new GCNN architecture that, coupled with a suited training procedure, makes it easier for the graph convolution layers to learn a mapping similar to the one of WL. Our contribution has two main components: a GCNN architecture suited for learning features similar to the ones of WL kernel, and a training procedure that enforces such features to be learned. We discuss each component in the following subsections.

4.1 Funnel GCNN Architecture

We modify the DGCNN architecture (see Section 2) in three ways. First, considering also the results in [20], we decide to replace the *SortPooling* layer. While in [38] the *sum pooling* results to be the most expressive among all pooling operations, it does not necessarily carry the optimal inductive bias. For this reason, inspired by [4] and after preliminary experiments, we decide to substitute the sortpooling with a concatenation of *max pooling, average pooling, and sum pooling*, computed globally over all the nodes in a graph, and on a concatenation of the node representation learned by the different graph convolutions. Then, we apply a small multi-layer perceptron to map from the graph hidden representations to the target(s). Moreover, we modify the activation function of graph convolution layers from *hyperbolic tangent* to *leaky ReLU*, thus mitigating the vanishing gradient problem. We also add *batch normalization*, that has been shown to further help the training process. We tested differentiable graph pooling layers [39, 4], but on all the considered datasets removing the pooling layers actually improved the performance, so we do not consider them in our architecture. However, in a future work, we plan to search more complex architectures including pooling. Finally, we notice that with DGCNN, and similarly with other graph neural network architectures, the number of graph convolutional filters used in the different layers is kept constant. This is in contrast with what happens with WL kernel, where the number of different features increases with the number of iterations. This observation holds with images as well, where it is common practice to increase the number of filters when increasing depth. Thus, we think it is essential to increase the number of convolution filters when stacking multiple graph convolutional layers. The resulting architecture (Funnel GCNN or FGCN) is simple, but it reflects the operations performed by the WL kernel and follows the complexity of its feature space having a number of convolution filters per layer that grows with depth.

4.2 Training GCNNs with Kernels

When dealing with machine learning on graphs, kernel methods are among the methods of choice, due to the theoretical guarantees they provide, and to the many efficient and effective kernel functions proposed in literature in the last years. Such kernels are known to provide a good representation for the input graphs, showing state-of-the-art results when coupled with linear models (in the feature space) such as SVM. We recall that, on the one hand, kernels generally compute the representations only implicitly (since the feature space can be very high - or even infinite- dimensional). On the other hand, given the relatively small size of graph datasets available online (some thousand examples, compared to the millions of examples usually required for training very deep neural networks), over-parametrized methods such as graph neural networks are hard to train, providing often solutions that do not generalize well. In this section, we present our method to incorporate information from graph kernels in the training of GCNNs, in order to obtain networks with high and predictable predictive performance. Our approach is tailored to the WL kernel. However, we also present an alternative (that shows comparable results) that is general in the sense that any kernel or vector embedding can be used, possibly exploiting the techniques in [14]. We limit ourselves to the WL kernel because: (i) finding embeddings hand-crafted by experts is difficult; (ii) it works well on tasks involving molecules, often performing better or comparably to state-of-the-art GCNN architectures; (iii) its complexity is linear in the size of the graphs; (iv) we can have a direct comparison to the technique proposed in [19].

4.2.1 Kernel-based Multi-Task Learning

As discussed in Section 2, the WL kernel provides an explicit graph embedding that is not task-specific. These embeddings are usually very high-dimensional and sparse. However, techniques for approximating these representations by hash functions have been proposed [21, 14]. The resulting graph embedding is dense and relatively low-dimensional. It is possible to exploit this explicit embedding to use graph kernels to drive the learning process of GCNNs in a different way compared to the one described in [19]. Specifically, we can resort

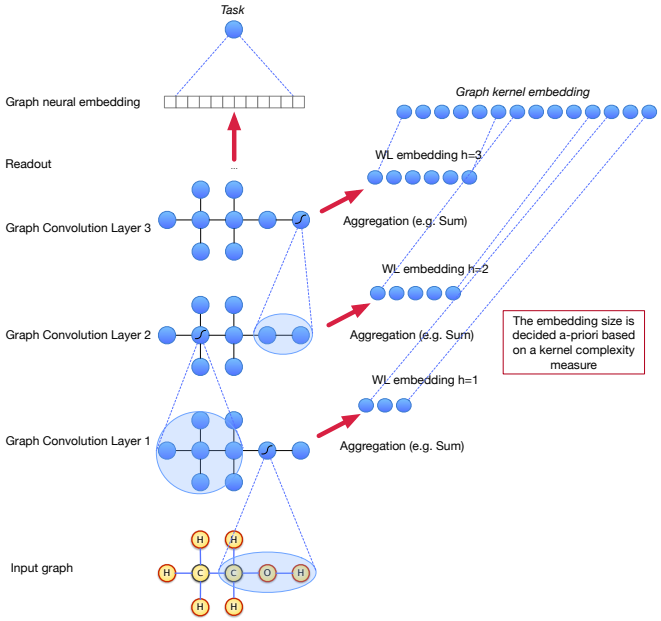

Figure 1. Layer-wise Multi-task training (LMT): WL kernel features at iteration i are provided as additional target for convolutional layers at depth i.

to multi-task learning (MTL) techniques and define a secondary output for the GCNN, that will try to reconstruct the kernel embedding in addition to the target y values of the considered task. In this way, we drive the representation learned by the GCNN to incorporate information about the kernel embedding. MTL is known to be beneficial for the learning process, in which it may act both as a representation bias, favoring representations that work well for all the considered tasks, as well as a regularizer.

For the sake of presentation, let us first introduce a simpler multi-task method compared to the one we adopt. Later, this method will be tested in our ablation study. Let us define two tasks. The first task is the original one, where the corresponding training data are $T_0 = D = \{(g^{(i)}, y^{(i)}), i = 1, \ldots, m\}$. We define the second task using a set of unlabeled graphs, $U = \{g^{(i)} \mid i = 1, \ldots, q\}$, possibly $U \supseteq D$. The second task is defined as $T_1 = \{(g^{(i)}, \phi(g^{(i)}))|g^{(i)} \in U\}$ where $\phi(\cdot)$ is the (possibly approximated) explicit feature representation coming from a graph kernel, or possibly any graph embedding. To limit the size of $\phi(\cdot)$ when using the WL kernel, we use hashing techniques [21] where we have an hyper-parameter controlling the resulting dimension. For many graph kernels, including the WL kernel, the computation of ϕ scales linearly in the size of the graph. In principle, training could alternate on the two tasks, effectively providing more training examples for the shared layers. Another option would be to jointly train the network on the two tasks, thus reducing MTL to multi-label learning or multi-output regression. We consider this second option because it is more efficient. During training, we just ignore the output of task T_0 if an example is in U and not in D.

We refer to this method as Multi-Task FGCNN, or MT-FGCNN for short, and we summarize it in Figure 2 in the supplementary material.

With many graph kernels, including WL, it is possible to group the features in buckets according to their structural complexity. For instance, features of WL can be split according to the WL iterations that generated them. In this way, we obtain not one, but a number of explicit representations equal to the number of WL iterations considered (the hyper-parameter h). The original $\phi(\cdot)$ of WL can be obtained simply concatenating these representation. The question is how to split the total hash size, i.e. given that we would like to use φ hash entries in total for the kernel explicit representation, how can we define a principled way to distribute them among features of different complexity? In [1], a complexity measure for (graph) kernels is defined, and it is shown that higher WL iterations correspond to more complex features. In fact, the number of features generated by *local* graph kernels, such as WL, increases with the iteration number, i.e. with the size of the features, as reflected by the kernel complexity. We decided to follow an approach in which we compute the complexity of the WL kernels at different iterations i, up to a maximum value h, that corresponds to the number of graph convolutional layers, say c_i^{WL}. Then we can compute the hash size η_i^{WL} for features at layer i as $\eta_i^{\text{WL}} = \frac{c_i^{\text{WL}}}{\sum_{j=1}^{h} c_j^{\text{WL}}} * \varphi$. We propose to adopt the resulting embeddings as secondary outputs of FGCNN, one for each graph convolution layer, so to drive the learnt representation toward points that incorporate (or encode) the information provided by the kernel.

We refer to this method as Layer-wise Multi-Task FGCNN, or LMT-FGCNN for short, summarizing it in Figure 1.

More formally, let us fix the maximum value h of the WL kernel, i.e. the number of WL iterations. Let us now define $h + 1$ tasks. The first task is the original one, where the corresponding training data are $T_0 = D = \{(g^{(i)}, y^{(i)}), i = 1, \ldots, m\}$. We define h tasks, one for each WL iteration, as

$$T_1 = \{(g^{(i)}, \phi^{(1)}(g^{(i)}))|g^{(i)} \in D\}, \ldots,$$
$$T_{h+1} = \{(g^{(i)}, \phi^{(h)}(g^{(i)}))|g^{(i)} \in D\}$$

where $\phi^{(i)}(\cdot)$ is the (approximated) explicit feature representation coming from a (hashed) WL kernel at iteration i.

Kernel approximation bounds With LMT and MT, the GCNN is approximating the explicit mapping provided by the WL kernel. Thus, the representation learned by the network can be seen as an approximated WL feature space, with the important note that the difference in representation may actually be beneficial since it is learned via back-propagation from the target of T_0. The network is thus learning a representation similar to the one of WL, but more aligned with the target. It is interesting to analyze the difference in the hypothesis $hyp(\cdot)$ obtained training an SVM using the original WL kernel, where $hyp(g)$ is the prediction for a graph g, and the one that can be obtained training an SVM on the representation learned by a GCNN using our approach, $hyp'(\cdot)$. We resort to a well-known approximation bound for the SVM from [6]. Let $\mathbf{K} \in \mathbb{R}^{t \times t}$ be the Gram matrix of WL kernel, and $\mathbf{K}'$ the one obtained considering the representation provided by a GCNN using our MTL approach. Let κ be the maximum value in $\mathbf{K}$ and $\mathbf{K}'$, and C be the SVM hyper-parameter. Then, for any graph g:

$$|hyp'(g) - hyp(g)| \leq \sqrt{2}\kappa^{\frac{3}{4}} C ||\mathbf{K}' - \mathbf{K}||_2^{\frac{1}{4}} \left[1 + \left[\frac{||\mathbf{K}' - \mathbf{K}||_2}{4\kappa}\right]^{\frac{1}{4}}\right].$$

Since we are approximating the explicit kernel feature space by the GCNN, we can easily compute a bound on $||\mathbf{K}' - \mathbf{K}||_2$ (proof in supplementary material) as follows.

Theorem 1 *Let $\delta_\pi(i)$ be the maximum approximation error for feature $\pi \in \{1, \ldots, \varphi\}$ in graph $g^{(i)}$, and let $\hat{\Delta} \geq \delta_\pi(i), \forall \pi \in \{1, \ldots, \varphi\}, i \in \{1, \ldots, m\}$. Then $||\mathbf{K}' - \mathbf{K}||_2 \leq 2m\hat{\Delta}\sqrt{\kappa} + m\varphi\hat{\Delta}^2$.*

Proof. We can rewrite $||\mathbf{K}' - \mathbf{K}||_2 = \sqrt{\sum_{i,j=1}^{m}(k'_{i,j} - k_{i,j})^2}$

$$= \sqrt{\sum_{i,j=1}^{m}\Big(\langle\phi(i) + \delta(i), \phi(j) + \delta(j)\rangle - \langle\phi(i), \phi(j)\rangle\Big)^2}$$

$$= \Big\{ \sum_{i,j=1}^{m} \Big[\sum_{\pi=1}^{\varphi} \Big(\phi_\pi(i) + \delta_\pi(i) \Big) \Big(\phi_\pi(j) + \delta_\pi(j) \Big) - \sum_{\pi=1}^{\varphi} \Big(\phi_\pi(i)\phi_\pi(j) \Big) \Big]^2 \Big\}^{1/2}$$

$$= \Big\{ \sum_{i,j=1}^{m} \Big[\sum_{d=1}^{\varphi} \Big(\phi_\pi(i)\phi_\pi(j) + \phi_\pi(i)\delta_\pi(j) + \phi_\pi(j)\delta_\pi(i) + \delta_\pi(i)\delta_\pi(j) - \phi_\pi(i)\phi_\pi(j) \Big) \Big]^2 \Big\}^{1/2},$$

where φ is the dimension of the graph embedding, and $\phi_\pi(i)$ and $\delta_\pi(i)$ are the π-th components of the vectors $\phi(i)$ and $\delta(i)$, respectively. Recalling that $\delta_\pi(i)$ is the maximum approximation error for feature π in example $g^{(i)}$, we can bound $\delta_\pi(i) \leq \hat{\Delta}, \forall \pi \in \{1,\dots,\varphi\}, i \in \{1,\dots,m\}$. Then $||\mathbf{K}' - \mathbf{K}||_2 \leq \sqrt{\sum_{i,j=1}^{m} \left(\sum_{\pi=1}^{\varphi} \left(\phi_\pi(i) + \phi_\pi(j) \right) \hat{\Delta} + \varphi \hat{\Delta}^2 \right)^2}$. We can bound the maximum 2-norm of each sample as $\max_{i \in \{1,\dots,m\}} ||\phi_i||_2 = \sqrt{\kappa}$. Thus $\sum_{\pi=1}^{\varphi} \left(\phi_\pi(i) + \phi_\pi(j) \right) \leq 2\sqrt{\kappa}$ and:

$$||\mathbf{K}' - \mathbf{K}||_2 \leq \sqrt{\sum_{i,j=1}^{m} \left(2\sqrt{\kappa}\hat{\Delta} + \varphi\hat{\Delta}^2 \right)^2} = \sqrt{m^2\hat{\Delta}^2(2\sqrt{\kappa} + \varphi\hat{\Delta})^2} = 2m\hat{\Delta}\sqrt{\kappa} + m\varphi\hat{\Delta}^2.$$

■

Being $\hat{\Delta}$ the reconstruction error, it can in principle be made arbitrarily small by using a large enough GCNN, given its approximation capability concerning WL kernel [38]. Moreover, a version of the theorem holding in probability can be given when the hashing technique to reduce the dimensionality of the kernel's feature space is used. In fact, it is sufficient to incorporate in $\hat{\Delta}$ the approximation bound given from the hashing, that holds in probability as a special case of a Random Projection Kernel [21, 37]. This bound provides guarantees on the performance of a GCNN trained with our method, since it ensures that the network will learn a representation that is close to the kernel's one. However, it does not, per se, guarantee better generalization performances. Nonetheless, if the target representation is good (as it is for the WL kernel), hopefully the representation learnt by the network will also be good, if not better. In fact, since the network representation is learned from data, it is not just an approximation of the kernel representation, but it is tailored for the task at hand, thus possibly improving the predictive performance compared to the one generated by the kernel. We plan to study the properties of the learned representations in more depth in a future version of the paper.

Computational Complexity One of the strengths of our proposed method compared to the pre-training approach based on siamese networks presented in [19] is that it scales linearly in the number of examples. In fact, computing WL features (not the kernel matrix) has $O(|E|)$ complexity for each graph, and is linear in the number of graphs. This is the same complexity of the graph convolution operator adopted in GCNNs. Kernel features are computed only once before training. During training, the use of the hashing technique allows to define a fixed (thus constant) number of (aggregated) features, allowing to keep the complexity of each forward pass constant w.r.t the number of features. Thus, our training procedure does not increment the asymptotic complexity of the training procedure compared to the standard training of GCNNs. On the contrary, the approach in [19] has quadratic complexity in the number of examples in the training set. As for the test phase, we don't need to consider the kernel embedding output, so we don't have to perform any additional operation compared to the standard GCNNs test phase.

5 EXPERIMENTAL RESULTS

In this section, we describe our experimental evaluation of the proposed LMT-FGCNN approach. Our source code is publicly available at *omitted for double-blind review*.

Datasets We perform experiments on five bioinformatics datasets from [40] involving node-labeled graphs, namely MUTAG, PTC, NCI1, PROTEINS and D&D. In the first three datasets, each graph represents a chemical compound, where nodes are labeled with the atom type, and edges represent bonds between them. PROTEINS and D&D consist of proteins represented as graphs in which nodes represent amino acids and two nodes are connected by an edge if they are less than 6Å apart.

Implementation details We adopted *PyTorch* [28] and *PyTorch Geometric* [9] for our implementation. We trained the neural networks using stochastic gradient descent with adaptive learning rate and momentum (*adam*), validating the learning rate in $\{0.01, \dots, 0.0001\}$. For our pre-training approach we used *mean square error* (MSE) as loss function for the training of the siamese network (pre-training), and *cross-entropy* (CE) as the loss function for the supervised phase. For the multi-task approaches, we used MSE for the output reconstructing the graph kernel, and CE for the primary task output. We combined the two losses by sum. We fixed the number of pre-training epochs to 100 for MUTAG and PTC, to 15 for NCI1 and 50 for PROTEINS and D&D (due to time limitations). We then fine-tuned the pre-trained DGCNN as usual on the training dataset. We validated for the FGCNN architecture the number of neurons in the first GC layer in $s_1 \in \{64, 128, 256\}$. We then fix the number of neurons in the following GC layers as $s_2 = s_1 \cdot 2$ and $s_3 = s_1 \cdot 3$. For our experiments, we adopted a machine with 2 x Intel(R) Xeon(R) CPU E5-2630L v3, 192GB of RAM and a Nvidia Tesla V100.

Methods We compare the performance of the DGCNN architecture in [40] with FGCNN trained with our layer-wise multi-task based approach (LMT-DGCNN). We employ the WL kernel with h fixed to 3 and with a feature space hashed into a vector of size 2000 or 5000 (depending on the dataset). As some related work suggests, the hash size (after a certain value) does not influence much the performance of the kernel [32]. In general, the larger the hash size, the closer the kernel is to the non-hashed version, but the slower the training of the network becomes. We did not find evidence of significant performance changes increasing the hash sizes, so we decided to leave it relatively small so not to impact the efficiency of training. The neural networks are evaluated via a 10-fold cross-validation, in which we used eight folds for training, one fold for validation and one fold for testing. We repeat the whole experiment 10 times, and report in Table 1 the average accuracy and the standard deviation.

Results and analysis The top section of Table 1 reports the results of our first set of experiments. The first line reports the performance of the graph kernel we adopted, WL with 3 iterations. The second line of the table reports the results of DGCNN. The performance of our proposed method, LMT-FGCNN, are reported in the third line. We can see that our approach performs better than DGCNN in all the considered datasets. We assess the statistical significance of the improvements performing a 10×10 CV test with confidence level 95% (and 10 degrees of freedom) for each pair of methods on

Table 1. Summary of our experimental results. We report accuracy± standard deviation. We underline the best results for each architecture, and we report in bold the highest performance on each dataset. Results marked with * are significantly worse than the best performing method.

	Method/Dataset	MUTAG	PTC	NCI1	PROTEINS	D&D
	WL (h=3)	76.79*±3.17	57.48*±1.36	82.13±2.17	69.63*±1.22	73.64*±2.56
	DGCNN	82.48*±1.49	57.14*±2.19	72.97*±0.87	73.96*±0.41	78.09*±0.72
	LMT-FGCNN	**86.81**±1.75	59.04±0.94	**82.20**±0.54	**76.03**±0.68	**80.14**±0.76
Ablation study	FGCNN	84.49±1.90	58.82± 1.80	81.50±0.39	74.57*±0.80	77.47*±0.86
Ablation study	LMT-DGCNN	85.00±1.15	**59.39**±0.51	77.02*±0.48	74.61*±0.89	78.11*±0.61
Ablation study	MT-DGCNN	83.68±1.29	58.39±1.11	76.55*±0.40	74.42*±0.36	78.17*±0.57
Ablation study	MT-FGCNN	85.81±1.62	59.23±2.35	81.86±0.41	75.18±0.66	79.90±0.39
Ablation study	PT-DGCNN	85.38±1.47	58.48±1.92	75.20* ±0.87	75.19±0.42	78.38*±0.55

all datasets [12]. We find that our proposed LMT-FGCNN performs significantly better than DGCNN on all the considered datasets.

LMT-FGCNN performs also better than the WL kernel in all the considered datsets. The same statistical significance test shows that the difference with WL is significant in four out of five datasets (on NCI1, the improvement of LMT-FGCNN is not significant compared to WL). Note that these datasets have been extensively used for more than a decade, and obtaining this kind of results is extremely difficult.

Computational Times Training FGCNN on NCI1 for 100 epochs takes 1m36s, while MT-FGCNN and LMT-FGCNN take 28s for the kernel computation, and 1m45s and 2m06s for training, respectively.

5.1 Ablation Study

In this section, we study the contribution of each component of our proposed method to the increment in performance reported in the previous section. We start analyzing the performance of the FGCNN architecture trained with a standard procedure. Then, we study the effect of our training procedure. First, we apply the same LMT procedure to DGCNN. Then we compare LMT with the simpler MT procedure, that does not incorporate layer-wise supervision. Finally, we compare our LMT method to the approach presented in [19], that we refer as PT (pre-training). Since its computation is very expensive, we compute PT-DGCNN only.

Contribution of Funnel architecture In the fourth line of Table 1, we report the results of the FGCNN architecture trained with a standard procedure (i.e. no kernel information is provided). The performances of FGCNN are higher compared to DGCNN in four out of five datasets. However, they are consistently lower compared to LMT-FGCNN. The statistical test confirms that FGCNN is significantly worse than LMT-DGCNN on two datasets, and comparable in the other datasets. These results indicate that FGCNN is overall a better architecture compared to DGCNN, but that our results are not solely due to the change in architecture.

Contribution of multi-task training In order to understand if our LMT training procedure is effective also on other neural architectures, we apply it to the DGCNN network, obtaining LMT-DGCNN that is reported in the fifth line of Table 1. Comparing LMT-DGCNN with DGCNN, we see that the performances of the former are always higher compared to those of the latter. The significativity test shows that the difference is significant in MUTAG and NCI1 datasets. Since LMT-DGCNN incorporates information from the WL kernel, we should compare its results with WL as well. In four out of five datasets, LMT-DGCNN improves over the WL kernel. A notable exception is NCI1, where the WL kernel performs significantly better compared to LMT-DGCNN. Let us now try to understand what prevents LMT-DGCNN to reach the WL performance as happens with LMT-FGCNN. We investigate how much multi-task training and pre-training affect the representation learnt by the lower graph convolution layers of DGCNN. We extract an *intermediate* hidden representation of DGCNN, obtained after the GC layers. We then sum the representations of all the nodes, and we run an SVM classifier on this representation. If the training procedure alters the representation learned by the graph convolution layers, we expect the SVMs trained on the intermediate representations extracted from DGCNN and (L)MT or PT-DGCNN to exhibit different predictive performances. We omit these results for lack of space. Surprisingly, for the majority of the datasets, we find out that the performance of the intermediate representations of DGCNN with and without kernel supervision are really close. This actually follows the intuition that in DGCNN it is difficult for the gradient to pass through the sortpooling operation in the readout and the tanh nonlinearities in the graph convolutional layers.

Contribution of Layer-wise embeddings in Multi-Task training Let us analyze the effects of the layer-wise supervision in LMT, comparing it with the more straightforward MT (see Section 4.2.1). Rows six and seven of Table 1 report the performance of MT-DGCNN and MT-FGCNN, respectively. While the MT method improves over the standard training approach, MT-FGCNN has almost always slightly lower performance compared to LMT-FGCNN (with the exception of PTC, where the two methods are comparable). Similar considerations apply to MT-DGCNN when comparing to LMT-DCNN. Overall, when comparing LMT with MT, we see a slight advantage for the former. However, the MT method is more general, not requiring to assign a feature to a specific layer of the GCNN. This result suggests that, in principle, the MT variant of our multi-task training procedure can be effective also when a single embedding is given for each graph.

5.2 Comparing LMT vs Other Strategies

In this section, we compare our LMT method with the related pre-training approach from [19], that we refer as PT. Since PT has a very high computational complexity, we computed it on DGCNN only. The last row of Table 1 reports the results of PT-DGCNN. Comparing it to LMT-DGCNN, we see that the two approaches are in general comparable. Not surprisingly, the bigger difference between the two methods is on the challenging NCI1 dataset, where it looks like incor-

Table 2. Comparison of our proposed FGCNN and LMT-FGCNN with graph kernels and different GCNNs architectures. We report accuracy± standard deviation of ten runs of 10-fold CV. †: results are obtained with the same model selection procedure used for the other models, that is different compared to [38].

Method/Dataset	MUTAG	PTC	NCI1	PROTEINS	D&D
RW	79.17±2.07	55.91±0.32	>3 days	59.57±0.09	>3 days
PK	76.00±2.69	**59.50**±2.44	82.54±0.47	73.68±0.68	78.25±0.51
WL	84.11±1.91	57.97±2.49	**84.46**±0.45	74.68±0.49	78.34±0.62
PSCN	-	-	76.34±1.68	75.00±2.51	76.27±2.64
GIN†	84.68±1.82	57.80±0.86	71.14±1.71	71.89±1.41	72.60±2.14
DGCNN	82.48±1.49	57.14±2.19	72.97±0.92	73.96±0.41	78.09±0.72
LMT-FGCNN	**86.81**±1.75	59.04±0.94	82.20±0.54	**76.03**±0.68	**80.14**±0.76

porating kernel information during training is more beneficial than using it in a pre-training phase. While the predictive performnce of LMT and PT are in general comparable, their running times are not, with LMT not significantly increasing the computational complexity of the training stage compared to standard approaches.

5.3 Comparison with state-of-the-art graph kernels and graph neural networks

For a baseline comparison, we report from [40] the performance of four state-of-the-art graph kernels: the graphlet kernel (GK) [31], the random walk kernel (RW) [36], the propagation kernel (PK) [22], and the Weisfeiler-Lehman subtree kernel (WL) [30]. The hyper-parameters' values of different kernels are selected as follows: the height of WL and PK in $\{0, 1, 2, 3, 4, 5\}$, the bin width of PK to 0.001, the size of the graphlets in GK to 3 and the decay of RW to the largest power of 10 that is smaller than the reciprocal of the squared maximum node degree. We report also the results of the other recent GCNN architectures: PSCN [24], and Graph Isomorphism Network (GIN) [38]. For GIN, we re-run the experiments following our experimental setting. We considered the hyperparameter values reported in the original paper, and validated the number of neurons in $\{16, 32\}$ and the learning rate in $\{0.01, 0.001\}$. In Table 2 we report the results of our experiments. Our method (LMT-FGCNN) shows higher performance when compared to other GNNs in all the datasets. Considering graph kernels, it shows better (on MUTAG, PROTEINS and D&D datasets) or competitive performance (comparable accuracy with respect to PK on PTC, and third best accuracy on NCI1).

6 SEMI-SUPERVISED LEARNING

In this section, we show how it is possible to use our proposed pre-training method to incorporate information from unlabeled examples in the training procedure. We consider other datasets compared to the first set of experiments since it is reasonable to have the data in U coming from approximately the same distribution of D. For this reason, we consider three datasets from the National Cancer Institute: NCI1B, NCI33B, NCI41B [26, 27]. Each dataset consists of a set of chemical compounds tested for their activity against a specific type of tumor. Note that NCI1B is slightly different from NCI1 adopted in the previous experiments. Inputs are thus drug-like chemical compounds in all cases. In Table 3 we report our results. As expected FGCNN performs significantly better than DGCNN. In addition, on these datasets, when $U = D$ (+0, 3rd column) LMT-FGCNN improves over FGCNN. In the 4th column (+1), we report the performances when U is composed by the graphs in 2 datasets. E.g., in the 4th column of the 1st row, U = NCI1B ∪ NCI33B, and obviously D = NCI1B. In the last column, +2, U is the merge of 3 datasets. We can see a consistent trend for the performances to improve when we add more datasets to the training procedure. Notice also that the learnt models are more stable, since the standard deviations tend to decrease when adding more unlabeled data. We think that these experiments can pave the way for future approaches that exploit unlabeled data in the training procedure to improve the performance of learning algorithms.

Table 3. Performance improvements when considering additional unlabeled data during training.

Dataset/ Method	DGCNN	FGCNN	LMT-FGCNN		
			+0	+1	+2
NCI1B	72.92 ±0.56	79.27 ±0.70	81.01 ±0.56	81.19 ±0.46	82.07 ±0.21
NCI33B	75.00 ±0.42	81.75 ±0.67	81.81 ±0.20	82.60 ±0.39	82.69 ±0.56
NCI41B	70.94 ±0.53	78.30 ±0.67	79.02 ±0.17	79.10 ±0.40	79.54 ±0.15

7 CONCLUSIONS AND FUTURE WORKS

In this paper, we presented an architecture and a training procedure designed to incorporate information conveyed by a graph kernel in the learning process of graph convolutional networks. Our method exploits the explicit representation available for some widely adopted graph kernels. We experiment our proposal on five real-world graph classification tasks, obtaining state-of-the-art performance. We conduct an ablation study to understand the contribution of each component of our method to the final results. The strength of our approach is that it does not rely on external (labeled or unlabeled) data to improve the performance of GCNNs. Nonetheless, it makes possible to incorporate additional unlabelled data to the training set, in a semi-supervised fashion. Preliminary results show that this approach can further improve the predictive performance of GCNNs, opening the door to future works in this direction.

ACKNOWLEDGEMENTS

This work has been supported in part by the University of Padova, Department of Mathematics, DEEPer project, and by the German Research Foundation (DFG) grant [BA 2168/3-3] and Germany's Excellence Strategy (CIBSS-EXC-2189 ID 390939984). The authors

acknowledge the HPC resources of the University of Padova, Department of Mathematics, made available for conducting the research reported in this paper.

REFERENCES

[1] Fabio Aiolli, Michele Donini, Nicolò Navarin, and Alessandro Sperduti, 'Multiple graph-kernel learning', in *Proceedings - 2015 IEEE Symposium Series on Computational Intelligence, SSCI 2015*, (2016).

[2] James Atwood and Don Towsley, 'Diffusion-convolutional neural networks', in *NIPS*, pp. 1993–2001, (2016).

[3] D. Bacciu, A. Micheli, and A. Sperduti, 'Generative kernels for tree-structured data', *IEEE Transactions on Neural Networks and Learning Systems*, **29**(10), 4932–4946, (Oct 2018).

[4] Cătălina Cangea, Petar Veličković, Nikola Jovanović, Thomas Kipf, and Pietro Liò, 'Towards Sparse Hierarchical Graph Classifiers', in *NIPS Relational Representation Learning Workshop*, (2018).

[5] Rich Caruana, 'Algorithms and applications for multitask learning', in *ICML*, pp. 87–95, (1996).

[6] Corinna Cortes, M Mohri, and A Talwalkar, 'On the impact of kernel approximation on learning accuracy', in *AISTATS*, pp. 113–120, (2010).

[7] Giovanni Da San Martino, Nicolò Navarin, and Alessandro Sperduti, 'Ordered Decompositional DAG Kernels Enhancements', *Neurocomputing*, **192**, 92–103, (2016).

[8] Giovanni Da San Martino, Nicolò Navarin, and Alessandro Sperduti, 'Tree-Based Kernel for Graphs With Continuous Attributes', *IEEE Trans. Neural Netw. Learning Syst.*, (2017).

[9] M. Fey, J. E. Lenssen, F. Weichert, and H. Müller, 'SplineCNN: Fast geometric deep learning with continuous B-spline kernels', in *IEEE CVPR*, (2018).

[10] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl, 'Neural Message Passing for Quantum Chemistry', in *Proceedings of the 34th ICML*, pp. 1263–1272, (apr 2017).

[11] William L. Hamilton, Rex Ying, and Jure Leskovec, 'Representation learning on graphs: Methods and applications', *CoRR*, **abs/1709.05584**, (2017).

[12] Nathalie Japkowicz and Mohak Shah, *Evaluating learning algorithms: A classification perspective*, volume 9780521196, 2011.

[13] Thomas N. Kipf and Max Welling, 'Semi-Supervised Classification with Graph Convolutional Networks', in *ICLR*, pp. 1–14, (2017).

[14] Nils M. Kriege, Marion Neumann, Christopher Morris, Kristian Kersting, and Petra Mutzel, 'A Unifying View of Explicit and Implicit Feature Maps for Structured Data: Systematic Studies of Graph Kernels', in *ICDM*, volume 2014, (2017).

[15] Tao Lei, Wengong Jin, Regina Barzilay, and Tommi Jaakkola, 'Deriving Neural Architectures from Sequence and Graph Kernels', in *ICML*, (2017).

[16] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel, 'Gated Graph Sequence Neural Networks', in *ICLR*, (2016).

[17] Alessio Micheli, 'Neural network for graphs: A contextual constructive approach', *IEEE Transactions on Neural Networks*, **20**(3), 498–511, (2009).

[18] Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe, 'Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks', in *AAAI*, (oct 2019).

[19] Nicolò Navarin, Dinh Van Tran, and Alessandro Sperduti, 'Pre-training Graph Neural Networks with Kernels', in *Machine Learning for Molecules and Materials NIPS Workshop*, (2018).

[20] Nicolò Navarin, Dinh Van Tran, and Alessandro Sperduti, 'Universal Readout for Graph Convolutional Neural Networks', in *IJCNN*, Budapest, Hungary, (2019).

[21] Nicolò Navarin, Giovanni Da San Martino, and Alessandro Sperduti, 'Extreme graph kernels for online learning on a memory budget', in *2018 IJCNN*, pp. 1–8, (July 2018).

[22] Marion Neumann, Novi Patricia, Roman Garnett, and Kristian Kersting, 'Efficient graph kernels by randomization', in *ECML-PKDD*, pp. 378–393. Springer, (2012).

[23] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov, 'Learning convolutional neural networks for graphs', in *ICML*, pp. 2014–2023, (2016).

[24] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov, 'Learning Convolutional Neural Networks for Graphs', in *ICML*, (2016).

[25] Giannis Nikolentzos, Polykarpos Meladianos, Antoine Jean-Pierre Tixier, Konstantinos Skianis, and Michalis Vazirgiannis, 'Kernel Graph Convolutional Neural Networks', in *ICANN*, pp. 22–32, (2018).

[26] Shirui Pan, Jia Wu, Xingquan Zhu, Guodong Long, and Chengqi Zhang, 'Finding the best not the most: Regularized loss minimization subgraph selection for graph classification', *Pattern Recognition*, **48**(11), 3783–3796, (2015).

[27] Shirui Pan, Jia Wu, Xingquan Zhu, Guodong Long, and Chengqi Zhang, 'Task Sensitive Feature Exploration and Learning for Multitask Graph Classification', *IEEE Transactions on Cybernetics*, **47**(3), 744–758, (2017).

[28] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer, 'Automatic differentiation in pytorch', in *NIPS-W*, (2017).

[29] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Ah Chung, Markus Hagenbuchner, and Gabriele Monfardini, 'The Graph Neural Network Model', *IEEE Transactions on Neural Networks*, **20**(1), 61–80, (2009).

[30] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt, 'Weisfeiler-Lehman Graph Kernels', *JMLR*, 2539–2561, (2011).

[31] Nino Shervashidze, S.V.N. Vishwanathan, Tobias H Petri, Kurt Mehlhorn, and Karsten M Borgwardt, 'Efficient graphlet kernels for large graph comparison', in *AISTATS*, pp. 488–495, (2009).

[32] Qinfeng Shi, James Petterson, Gideon Dror, John Langford, Alex Smola, and S V N Vishwanathan, 'Hash Kernels for Structured Data', *Journal of Machine Learning Research*, **10**, 2615–2637, (nov 2009).

[33] Alessandro Sperduti and Antonina Starita, 'Supervised neural networks for the classification of structures', *IEEE Trans. Neural Networks*, **8**(3), 714–735, (1997).

[34] Dinh V. Tran, Nicolò Navarin, and Alessandro Sperduti, 'On Filter Size in Graph Convolutional Networks', in *IEEE SSCI*, Bengaluru, India, (2018).

[35] Laurens Van Der Maaten, 'Learning discriminative fisher kernels.', in *ICML*, volume 11, pp. 217–224, (2011).

[36] S.V.N. Vishwanathan, Nicol N Schraudolph, Risi Kondor, and Karsten M Borgwardt, 'Graph kernels', *Journal of Machine Learning Research*, **11**(Apr), 1201–1242, (2010).

[37] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg, 'Feature hashing for large scale multitask learning', in *ICML*, New York, New York, USA, (2009). ACM Press.

[38] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka, 'How Powerful are Graph Neural Networks?', in *ICLR*, (2019).

[39] Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec, 'Hierarchical Graph Representation Learning with Differentiable Pooling', in *NIPS*, (2018).

[40] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, 'An End-to-End Deep Learning Architecture for Graph Classification', in *AAAI*, (2018).