Graph matching as a graph convolution operator for graph neural networks

Maxime Martineau, Romain Raveaux, Donatello Conte, Gilles Venturini

Graph matching as a graph convolution operator for graph neural networks

Maxime Martineau[1], Romain Raveaux[1], Donatello Conte[1], and Gilles Venturini[1]

[1]Université de Tours, Laboratoire d'Informatique Fondamentale et Appliquée de Tours (LIFAT - EA 6300), 64 Avenue Jean Portalis, 37000 Tours, France

June 22, 2021

Abstract

Convolutional neural networks (CNNs), in a few decades, have outperformed the existing state of the art methods in classification context. However, in the way they were formalised, CNNs are bound to operate on euclidean spaces. Indeed, convolution is a signal operation that are defined on euclidean spaces. This has restricted deep learning main use to euclidean-defined data such as sound or image. And yet, numerous computer application fields (among which network analysis, computational social science, chemo-informatics or computer graphics) induce non-euclideanly defined data such as graphs, networks or manifolds. In this paper we propose a new convolution neural network architecture, defined directly into graph space. The convolution operator is defined in graph domain thanks to a graph matching procedure between the input signal and a filter. We show its usability in a back-propagation context. Experimental results show that our model performance is at state of the art level on simple tasks. It shows robustness with respect to graph domain changes and improvement with respect to other euclidean and non-euclidean convolutional architectures.

1 Introduction

Graphs are frequently used in various fields of computer science, since they constitute a universal modeling tool which allows the description of structured data. The handled objects and their relations are described in a single and human-readable formalism. Hence, tools for graphs supervised classification and graph mining are required in many applications such as pattern recognition [19], chemical components analysis [9], structured data retrieval [18]. Graph classification can be operated on graph space. For instance, it can be done via a k-nearest neighbour setting using graph matching [19, 4, 15, 14]. Alternatively, graph classification can also be achieved through vector-based representation of graph and classic pattern recognition techniques (vector space graph classification) [19].

1.1 Euclidean and geometric deep learning

Deep learning has achieved a remarkable performance breakthrough in several fields, most notably in speech recognition, natural language processing, and computer vision. In particular, convolutional neural network (CNN) architectures currently produce state-of-the-art performance on a variety of image analysis tasks such as object detection and recognition. Most of deep learning research has so far focused on dealing with 1D, 2D, or 3D Euclidean structured data such as acoustic signals, images, or videos.

Recently, there has been an increasing interest in geometric deep learning, attempting to generalize deep learning methods to non-Euclidean structured data such as graphs and manifolds, with a variety of applications from the domains of network analysis, computational social science, or computer graphics.

1.2 Graph Neural Networks

Among geometric deep learning methods are graph neural networks. These neural networks often try to apply convolution to graphs so that it mimics classical convolutional neural networks. Defining on graph space is not trivial. There is indeed no straightforward definition. However, one can identify two families of definitions in the existing literature [5]. The first family (spectral approaches among which [7, 11]) relies on the convolution theorem. This theorem states that the convolution operator on the spatial domain is equivalent to the product operator on the frequency domain. Although this theorem was only proven on euclidean spaces, spectral approaches postulate its validity on the graph space.Such approaches have two main limitations. The first one is their sensitivity to topological variations: a slight deformation of the graph structure changes the resulting convolution signal drastically. The latter is that there is no Fast Fourier Transform on the graph space: as previously stated, accessing the graph frequency domain relies on matrix diagonalization and therefore inversion. Inverting a matrix is a costly operation. The second family of approaches

(the spatial ones, [22, 2, 17, 24, 16]) tries to come up with analogies of the original convolution definition. However, existing approaches don't rely solely on the graph domain as they use approximations of graphs (through aggregations [10], expression as manifolds [16], ...).

In this paper, we propose a graph convolution operator which operates solely on graph space. This is made possible by using graph matching to define local convolutional operation. By doing so, we try to establish a link between two scientific communities who respectively work on graphs and deep learning. More specifically, we define graph-based computations using operators from the graph matching litterature in a deep learning (neural network) framework.

2 State of the Art

Every graph neural network layer can then be written as a non-linear function: $H^{(l+1)} = f(H^{(l)}, A)$.

As an example, let's consider the following very simple form of a layer-wise propagation rule: $f(H^{(l)}, A) = \sigma\left(D^{-1}AH^{(l)}W^{(l)}\right)$. $\sigma(.)$ is a non-linear activation function like the ReLU. Multiplying the input with $D^{-1}A$ now corresponds to taking the average of neighboring node features from the layer l. It is also called in the literature "average neighbor messages" and it acts like passing average node features from one layer to another. In [11], a better (symetric) normalization of the adjacency matrix is proposed i.e. $D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$. A per-neighbor normalization is performed instead of simple average, normalization varies across neighbors: $f(H^{(l)}, A) = \sigma\left(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$ with $\hat{A} = A + I$, where I is the identity matrix and $\hat{D}$ is the diagonal node degree matrix of $\hat{A}$.

More operations have been investigated in the literature [17]. A complete family of operations can be used :

I: this identity operator does not consider the structure of the graph and neither provide any aggregation. Used alone this operator makes the GNN a composition of $|V|$ MLP completly independent. One MLP for each node feature vector.

A: The adjacency operator gather information on the node neighborhood (1 hop).

D: $D = diag(A\mathbf{1})$. This degree operator gather information on the node degree. D is node degree matrix (a diagonal matrix). A_j: $A_j = min(1, A^{2^j})$. It encodes 2^j-hop neighborhoods of each node, and allow us to aggregate local information at different scales, which is useful in regular graphs.

U: U is matrix filled with ones. This average operator, which allows to broadcast information globally at each layer, thus giving the GNN the ability to recover average degrees, or more generally moments of local graph properties.

Let us denote $\mathcal{A} = \{I, D, A, A_1, \cdots, A_J, U\}$. A GNN layer is defined as : $f(H^{(l)}, \mathcal{A}) = \sigma\left(\sum_{B\in\mathcal{A}} BH^{(l)}W_B^{(l)}\right)$. $\Omega = \{W_1^{(l)}, \cdots, W_{|\mathcal{A}|}^{(l)}\}, W_B^{(l)} \in \mathbb{R}^{m_{(l)}\times m_{(l+1)}}$ are trainable parameters.

Key distinctions are in how different approaches aggregate messages. So far, proposals have aggregated the neighbor messages by taking their (weighted) average, but is it possible to do better? In [10], a GNN called GraphSAGE is proposed. The aggregation of neighbors information is more complex. The very general scheme of aggregation can written thanks to the function AGG: $H^{(l+1)} = \sigma\left(AGG(H^{(l)})W^{(l)}\right)$. Let us define $\mathcal{N}(u)$ is the set of nodes in the 1-hop neighborhood of node u.

$mean$: $AGG_u = \frac{1}{|\mathcal{N}(u)|}\sum_{v\in\mathcal{N}(u)} H_v^{(l)} \quad \forall u \in V \implies AGG = D^{-1}AH^{(l)}$.

max : $AGG_u = max(\{H_v^{(l)}, \quad \forall v \in \mathcal{N}(u)\}) \quad \forall u \in V$. Transform neighbor vectors into a matrix and apply a max pooling element-wise.

$LSTM$: $AGG_u = LSTM([H_v^{(l)}, \quad \forall v \in \pi(\mathcal{N}(u))]) \quad \forall u \in V$. Where π is a random permutation. The idea is to provide to the LSTM a sequence composed of neighbor embeddings. So the input sequence is composed of vectors. The sequence is randomly permuted by the function π.

In [16], the graph structure is locally embedded into a vector space. The distribution of local structures in the local space is estimated by a Gaussian Mixture Model. The AGG_u function is then expressed by a mixture of Gaussians. The Gaussian parameters are covariance matrix and mean vector and they are learnt during the training of the neural network.

A notable variant of GNN is graph attention networks (GAT), which was first proposed in [24]. This model includes the self attention mechanism to evaluate the individual importance of the adjacent nodes and therefore it can be applied to graph nodes having different degrees by specifying arbitrary weights to the neighbors [24]. For further reading, good surveys about graph neural networks have been published [28, 27, 25].

Deadlocks, contributions and motivations From the literature, two main deadlocks can be drawn. First, in many of the related works [11, 17, 24], edge features are not well considered. However, the edge information is of first interest to boosts the structural knowledge in the computation of the node embedding. Second, most of the aforementioned approaches do not take full advantage of the graph topology [16, 11]. The graph structure is locally embedded into a vector space (i.e. the tangent space at a given point of a riemannian manifold). In this paper, we propose CNN architectures that remain in the graph domain. Especially, we design a convolution operator onto graph space through the solution of a graph matching problem. The problem of graph matching under node and pair-wise constraints is fundamental to capture topological information. It takes into account the nodes and edge features along with their neighborhood structure. Therefore, graph matching-based convolution can release deadlocks related to edge information integration, domain changes sensitivity and Euclidean space projection. Graph matching can be seen as added local constraints in the machine learning problem. We promote a truly novel class of neural network architecture where layers contain a combinatorial optimization scheme that plays a fundamental role in the construction of the entire neural network architecture. Consequently, we highlight the interplay between machine

Table 1: Frequently used notations

Notation	Description
G_I	An input graph
G_F	A filter graph
g_I^i	Neighbourhood subgraph rooted at vertex i in I
i, j	Vertices in graph G_I
ij	An edge in graph G_I between i and j
a	A vertex in G_F
ab	An edge in G_F between a and b
μ	Labelling function for vertices
ζ	Labelling function for edges
G_F^W	A filter graph and its associated weights
$\mu(a)$	Vertex label of a
$\mu^W(a)$	Vertex label of a parametrized by W
$\|\Omega_{ij}\|$	Cardinality of Ω_{ij}
δ_x^y	Kronecker delta of x and y

learning and combinatorial optimization.

3 Graph Convolutional Neural Network

Frequently used notations are summarized in Table 1.

3.1 Graph matching

To define our convolution operator, we must define the graph matching function that will be pointwisely used.

Let G_1 and G_2 be attributed graphs: $G_1 = (V_1, E_1, \mu_1, \zeta_1)$ and $G_2 = (V_2, E_2, \mu_2, \zeta_2)$

$$\text{GMS}(G_1, G_2) = \max_y \quad s(G_1, G_2, y), \tag{1a}$$

$$\text{subject to} \quad y \in \{0,1\}^{n_1 n_2} \tag{1b}$$

$$\sum_{i=1}^{n_1} y_{i,a} = 1 \quad \forall a \in [1, \cdots, n_2] \tag{1c}$$

$$\sum_{a=1}^{n_2} y_{i,a} \leq 1 \quad \forall i \in [1, \cdots, n_1] \tag{1d}$$

$$|V_1| \geq |V_2| \tag{1e}$$

The similarity function s is defined as follows:

$$s(G_1, G_2, y) = \sum_{y_{ia}=1} s_V(i,a) + \sum_{y_{ia}=1} \sum_{y_{jb}=1} s_E(ij, ab) \tag{2a}$$

$$s_V(i,a) = \mu_1(i).\mu_2(a) \tag{2b}$$

$$s_V(i,\epsilon) = s_V(\epsilon,a) = 0 \tag{2c}$$

$$s_E(ij, ab) = \zeta_1(ij).\zeta_2(ab) \tag{2d}$$

$$s_E(ij, \epsilon\epsilon) = s_E(\epsilon\epsilon, ab) = 0 \tag{2e}$$

Let $\pi(G_1, G_2, e)$ denote an assignment of element (edge or vertex) $e \in V_1 \cup E_1$ to some element in $V_2 \cup E_2 \cup \{\epsilon, \epsilon\epsilon\}$:

$$\pi(G_1, G_2, i) = a \iff \exists a \in V_2 : y_{ia} = 1 \tag{3a}$$

$$\pi(G_1, G_2, i) = \epsilon \iff \forall a \in V_2 : y_{ia} = 0 \tag{3b}$$

$$\pi(G_1, G_2, ij) = ab \iff \exists ab \in E_2 : y_{ia} = 1 \wedge y_{jb} = 1 \tag{3c}$$

$$\pi(G_1, G_2, ij) = \epsilon\epsilon \iff \forall a, b \in V_2 : y_{ia} = 0 \vee y_{jb} = 0 \tag{3d}$$

The similarity function can be rewritten as follows:

$$s(G_1, G_2, y) = \sum_{i \in V_1} s_V(i, \pi(G_1, G_2, i)) + \sum_{ij \in E_1} s_E(ij, \pi(G_1, G_2, ij)) \tag{4a}$$

3.2 Graph convolution based on graph matching

Now that our matching operator is formulated, we can apply it over an input graph to compute the result of a convolution.

Let G_I and G_F be attributed graphs: $G_I = (V_I, E_I, \mu_I, \zeta_I)$ and $G_F = (V_F, E_F, \mu_F, \zeta_F)$. G_I and G_F are respectively referred to as the input graph and the filter graph.

3.2.1 Graph convolution operator $\odot$

The graph convolution operator is a function $\mathbb{G} \times \mathbb{G} \to \mathbb{G}$ and is defined as follows:

$$G_I \odot G_F = (V_I, E_I, \mu, \zeta) \tag{5a}$$

$$\text{with} \quad \mu : V_I \to \mathbb{R} \quad \text{such that} \quad \mu(i) = \text{GMS}(g_I^i, G_F) \tag{5b}$$

$$\zeta : E_I \to \mathbb{R} \quad \text{such that} \quad \zeta(ij) = \text{score}(ij, G_I, G_F) \tag{5c}$$

where g_I^i and *score* are defined as follows.

3.2.2 Vertex neighbourhood graph (l-hops)

g_I^i is defining the neighbourhood (which is a subgraph) for vertex i in G_I:

$$g_I^i = (N_I^l[i], E_I^i, \mu_I, \zeta_I) \tag{6a}$$

$$\text{with} \quad N_I^l[i] \text{ the } l\text{-hops closed neighbourhood of } i \text{ in } G_I \tag{6b}$$

$$\text{and} \quad E_I^i = \{kl \in E_I \quad \text{s.t.} \quad k, l \in N_I^l[i]\} \tag{6c}$$

3.2.3 Edge attribute in convolved graph

score is a function mapping an edge to its matching score in the found GMS. The problem is that it might be assigned multiple times:

$$\text{let} \quad \Omega_{ij} = \{g_I^k \quad \forall k \in V_I : ij \in g_I^k\} \quad \forall ij \in E_I \tag{7}$$

Ω_{ij} potentially contains more than one element. Therefore, score can be defined as follows:

$$\text{score}(ij, G_I, G_F) = \theta\left(\{s_E(ij, \pi(g_I, G_F, ij)) \quad \forall g_I \in \Omega_{ij}\}\right) \tag{8a}$$

$$\text{with} \quad \theta : \text{some statistical estimator (max or avg)} \tag{8b}$$

3.3 Convolution layer

Now that the convolution operator is defined, it is possible to use it as a base to build a convolution layer. This layer can be included in a graph neural network.

3.3.1 Graph convolution filter: the filter graph

A graph convolution filter is an attributed graph G_F^W. Its role is analogous to that of a vanilla CNN kernel: it modifies the output and gets modified through backpropagation. Every attribute function is parametrized with respect to a weight vector $W \in \mathbb{R}^{|V|+|E|}$.

$$G_F^W = (V_F, E_F, \mu_F^W, \zeta_F^W) \tag{9a}$$

$$\text{with} \quad \mu_F^W(a) = W_a \tag{9b}$$

$$\zeta_F^W(ab) = W_{ab} \tag{9c}$$

3.3.2 Graph convolution layer

A convolution layer is a set of convolution filters $\{G_F^p\}_{1 \leq p \leq n}$ applied on a same input graph G_I. The output of the layer consists of all filters results (analogous to euclidean convolution feature map) stacked up.

Let u be the output function of the layer s.t.:

$$u : \mathbb{G} \to \mathbb{G} \quad u(G_I) = \psi(\{u_p(G_I)\}_{1 \leq p \leq n}) \tag{10a}$$

$$\text{with} \quad \psi : \mathbb{G}^n \to \mathbb{G} \quad \psi(\{u_p(G_I)\}_{1 \leq p \leq n}) = (V_I, E_I, M, Z) \tag{10b}$$

$$M : V_I \to \mathbb{R}^n \quad (M(i))^p = \mu_F^p(i) \tag{10c}$$

$$Z : E_I \to \mathbb{R}^n \quad (Z(ij))^p = \zeta_F^p(ij) \tag{10d}$$

$$n \text{ the number of filters} \tag{10e}$$

ψ function keeps only a single graph structure and concatenates each vertex/edge attribute. The output function of the layer is a graph with same topology as G_I but with attributes as vectors composed by attributes of every filters outputs.

Graph convolution computation can be seen as a step-by-step process. The first step is neighbourhood extraction: for each vertice i in G_I (the input graph), the neighbourhood graph g^i is extracted. It is composed of every neighbour of i in a given range (it can be 1-hop away but also n-hops away). g_i and G_F (the filter graph) are matched. The matching score $\text{GMS}(g^i, G_F)$ becomes the output of the convolution at i.

3.4 About graph matching differentiation

In our method, a differentiation of the convolution operator is proposed. This differentiation does not take into account the dependencies between the optimal graph matching $\hat{y}$ and the variables $\{\mu_I(k)\}_{k\in E_I}$ and $\{\zeta_I(kl)\}_{kl\in V_I}$. As these variables are used to calculate the possible matchings, it is trivial to conclude such dependencies exist. Nevertheless, the matching solver in use (see Subsection 3.8) is not differentiable, at least a priori. We therefore assumed $\hat{y}$ as a constant in the gradient calculus with respect to these variables.

3.5 A "no edge matching" version of the graph convolution layer

This section presents a degraded model. It ignores topology at a local level by not matching edges. It therefore reduces the graph matching problem to a node assignment problem inside a given neighborhood. One concern on this simplification could be that we do not take advantage of the graphs topology and edges. However, topology and edge information is used when computing vertices neighbourhoods. If no edge exist between nodes i and j, i won't be included in g_1^j and neither will j in g_1^i. In any case, edge attributes are never used. Additionally, this model has lower time complexity as edge information is not taken into account (see details in Subsection 3.8.)

Used graphs are 3-uplets (V, E, μ) and the similarity function is simplified as follows:

$$s(G_1, G_2, y) = \sum_{y_{ia}=1} s_V(i, a) \tag{11a}$$

$$s_V(i, a) = \mu_1(i).\mu_2(a) \tag{11b}$$

As a consequence of the edge attributes deletion in the filter graph, its parameter becomes vector $W \in \mathbb{R}^{|V|}$ (as many parameters as vertex). The filter is defined as follows:

$$G_F^W = (V_F, E_F, \mu_F^W) \tag{12a}$$

$$\text{with} \quad \mu_F^W(a) = W_a \tag{12b}$$

The output function of the filter $u : \mathbb{G} \to \mathbb{G}$ is defined as follows:

$$u(G_I) = G_I \odot G_F^W \tag{13a}$$
$$= (V_I, E_I, \mu) \tag{13b}$$

3.6 Graph pooling

As in euclidean convolutional neural nets, we want to implement not only convolutional layers but also pooling/downsampling layers. In the existing literature, downsampling is view as graph coarsening [5]. A recurrent graph coarsening algorithm choice seems to be Graclus [8] (used in [16, 7]). We propose to use a community detection algorithm (Louvain method [3]) as the base of our graph pooling layer. Louvain method deals with weighted graphs. In our case, edge weights are computed by scalar products of involved vertices. This choice is brought by the following intuition: the higher nodes attributes scalar product get, the more these vertices probabilities to fall in the same cluster increases (because a higher scalar product implies vector similarity).

3.7 Hyperparameterization

As in any neural network, graph neural networks have parameters that won't be optimized from gradient descent. The first one is the graph filter (its number of nodes and adjacency matrix). The number of nodes in the graph filter is analogous to the size of a classic convolution kernel (for example, 3×3 kernel filter is equivalent to a 9 nodes filter graph with grid-like adjacency). The second hyperparameter is the size of extracted neighbourhoods graphs which is the maximum node distance in a given node neighbourhood. A 2-hop-sized neighbourhoods will include nodes that can be reached from the origin node in two hops or less. These hyperparameters could be optimized through grid or random search. However, to restrain our study, we will consider the following postulate: a graph filter should be congruent with the neighbourhoods. In other words, filters and neighbourhoods should have equal sizes and identical topologies as much as possible. This postulate comes from classic graph convolution where each kernel coefficient is matched with one and only one image coefficient. As we only experimented with the "no edge-matching" model, the filters topologies weren't to be defined. However we set the filters size to the average neighbourhood size in the dataset.

3.8 Choosing the graph matching solver

The algorithm for solving the graph matching problem is a critical element for the model. The first reason is that it is potentially the highest in complexity since graph matching problems are up to NP-hard. Additionally, graph matching is solved as many times as there are vertices in the input graph (the size of

every problem to solve being that of every vertex neighbourhood). We opted for a bipartite (BP) graph matching algorithm [21]. Complexity of such an algorithm is among the lowest (polynomial time) for solving error-tolerant graph matching problems suboptimally. Bipartite graph matching algorithm reduces graph matching to vertex matching by embedding an estimation for edge costs in the vertex costs. This edge cost estimation is computed by solving an edge-assignment problem for every node-matching possibility. Therefore, BP has to solve as many matching problems as there are edge-costs.

We used a variant of BP called Square Fast BP [23] where the cost matrix for vertex matching is of size $\max(|g_I|, |G_F|) \times \max(|g_I|, |G_F|)$ with $|G_F|$ and $|g_I|$ being number of vertices in filter graph G_F and neighbourhood graph g_I. Assuming both neighbourhood and filter graphs are complete, a matching problem complexity is $O(\max(|g_I|, |G_F|)^3)$. As a consequence, worst case complexity with fast bipartite matching is $O(\max(|g_I|, |G_F|)^5)$. Some preliminary experiments showed impracticable computation time of the full model. As a first workaround, the experimental part of this paper will focus on "no edge matching" model. This workaround allowed to keep processing to an acceptable level (that is suitable for small classification experiments). Edge cost estimation by edge matching is no longer required. The simplified model has $O(\max(|g_I|, |G_F|)^3)$ as pointwise complexity.

We should stress that the choice of the graph matching solver is critical because it defines our model capabilities of handling rich graph data: if the GM operator can handle oriented attributed graphs, so can our model.

4 Experimental work

In this section, we test the model according to several parameters. We want to test our model with a simple classification task on MNIST digit images. Code for running the model can be found at `https://github.com/prafiny/graphconv`

4.1 Baselines

Our approach was compared with three other approaches: a°) a Vanilla CNN layer, b°) MoNET [16] a mixture model graph CNN and GraphSAGE [10]. Same network topology was used for all approaches. It consists of classical ConvPool blocks linearly connected. Figure 1 shows the exact network structure in use. In case of graph convolution, $n \times n$ convolution filters equivalents are n^2 nodes filters and 2×2 pooling becomes 4 nodes pooling. n is set depending on average graph connectivity in a given dataset: if the average number of neighbours in a given dataset is 9, $n = 9$. The last layer is a global pooling one. As in the euclidean case, it consists in aggregating each filter feature map in one scalar value. In our case, feature maps are aggregated by taking its average value.

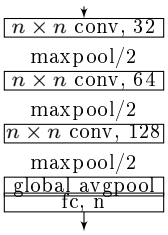

Figure 1: Network structure used for graph convolution experiments

4.2 Data

4.2.1 MNIST

Quantitative experiments in this section are operated on digit images of MNIST dataset [13]. We chose this dataset as this was in use in the graph convolution literature. MNIST is a good "hello world" machine learning (ML) dataset. MNIST helps at quickly iterating on the learning model. Performance information gathered from experiments on MNIST can be great for judging how the model might perform on much harder and larger datasets like ImageNet. In addition to the original MNIST dataset, a rotated version was used [12]. To compare results with MNIST-rotated, MNIST-reduced has been extracted from MNIST to match MNIST-rotated cardinalities (see Table 2). Note that the test set of MNIST-reduced is larger than the training set by a factor 5, thus the generalization ability is better assessed.

Table 2: Different MNIST-based graph datasets

Dataset	**Training set**	**Validation set**	**Testing set**
MNIST-original	48 000	12 000	10 000
MNIST-rotated	10 000	2 000	50 000
MNIST-reduced			
MNIST-mixed			

Lastly, to test rotation invariance, a third MNIST-based dataset was added: MNIST-mixed. It was generated by combining MNIST-reduced train and validation sets and MNIST-rotated test set. It is design so that the models are trained on rotation-free images but tested on rotated images. As MNIST is an image dataset, a graph-based representation of images has to be chosen. Representations used in [16] are superpixels graphs and grid graphs. We used $\frac{1}{4}$ grids (28×28 images resized to 14×14) and generated 75 superpixels Region Adjacency Graphs (RAG) using SLIC algorithm [1] with superpixel adjacency as edges (see Table 3). Sample graphs are depicted in Figure 2.

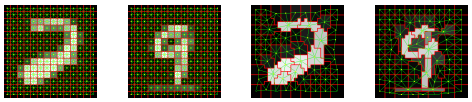

Figure 2: MNIST graphs. Left is $\frac{1}{4}$ grid, right is 75 superpixels RAG. Red symbolizes vertex frontiers and green shows edges.

Table 3: MNIST representations

Representation	**Nb nodes**	**Vertex attributes**	**Edge attributes**
$\frac{1}{4}$ grid	14^2	Pixel intensities	Relative polar coordinates
75 superpixels	75 (average)	Average superpixel intensities	

4.2.2 IAM graphs

Classification experiments have been conducted on graph-based data: IAM graph datasets Web, Mutagenicity and AIDS [20]. Mutagenicity and AIDS are a graph collection composed of molecular data: nodes and edges are respectively representing atoms and chemical bonds. The classification tasks for these two sets is to infer a given chemical property for each graph. Web graphs represent web pages in terms of text they contain: nodes are words occuring in a given page and edges represent context relationships between words. The classification task for Web is to identify each page category. Table 4 describes in further details every dataset.

Database	size (tr, va, te)	# classes	node labels	edge labels	$\overline{\lvert V \rvert}$	$\overline{\lvert E \rvert}$	max $\lvert V \rvert$	max $\lvert E \rvert$	balanced
AIDS	(250, 250, 1 500)	2	Chemical symbol	Valence	15.7	16.2	95	103	N
Mutagenicity	(1 500, 500, 2 337)	2	Chemical symbol	Valence	30.3	30.8	417	112	N
Web	(780, 780, 780)	20	Word and frequency	Section type	186.1	104.6	834	596	N

Table 4: Summary of graph data set characteristics, viz. the size of the training (tr), the validation (va) and the test set (te), the number of classes, the label alphabet of both nodes and edges, the average and maximum number of nodes and edges, whether the graphs are uniformely distributed over the classes or not (balanced) [20]

4.3 Parameterization

Following hyperparameters were set after preliminary tests were conducted: models are trained during 100 epochs using Adaptive Moment (Adam) gradient descent (learning rate 10^{-4}). Neighbourhood reach in use is 1-hop and filter size was set in accordance with average neighbourhood size (9 nodes).

4.4 Protocol

Following experiments were conducted:

Experiment 1 Models are tested on MNIST digit images classification task

Experiment 2 Models are tested on IAM graph datasets classification tasks

Experiment 3 Several neighbourhood connectivities are tested on our model (1 and 2 hops)

Experiment 4 Rotation invariance is investigated. Spatial information for our datasets is conveyed by edge attributes. In such a frame, as our "no edges" model ignores edge attributes, it is theoretically rotation-invariant. Experiment 3 aims at experimentally validating this claim. This is done by training models on unrotated images and testing on rotated ones. MNIST-mixed set is used to this end.

Experiment 5 Graph based methods are tested on regular grids and on irregular graphs (75 superpixels RAG) for testing sensitivity to domain changes

Experiment 6 A sample filter is visualized on some MNIST example images

As stated before and because of technical limitations, experiments involving MNIST datasets will focus on the two first MNIST classes (referred to as MNIST-2class)

4.5 Results

On MNIST-2class (Experiment 1), results are depicted in Table 5. Our model competes in a 3% margin with used baselines. On IAM Graph (Experiment 2), the results are reported in Table 6. They show that our method achieved similar or better results than baselines except on the Web database where GraphSAGE performed better. This exception can be explained by the fact that the connectivity of the Web database is the lowest among all the data sets. The web data set has more nodes than edges in average. GraphSAGE has specific parameters only defined for the node' features without considering the neighborhood. This can give a strong advantage in that case. Our model performed well on the IAM database due to the importance of node information during matching (in the contrary to MNIST experiments): when classifying an image, edge information is mainly used to orient the filter (as in classical CNN). In the contrary, it might be different on other types of data such as molecules or pages. Extending the neighbourhood size (Experiment 3) did not have any significant effect on performance (see Table 7). A possible reason is that experiments were performed with the “no edge-matching" model where edge attributes are not taken into account. Therefore, increasing the size of neighbourhoods provides only little information because this information is not spatially defined. The bigger the neighbourhood gets, the more critical spatiality becomes. On MNIST-mixed (Experiment 4), no performance loss was observed on testing for our method. This is especially visible on grid graphs results where only classic CNN and MoNet show a 10 percent loss. A trivial explanation of how is this invariance

Table 5: Recognition rates on MNIST 2class

Representation	**Dataset**	**CNN**		**MoNet**		**Ours**	
		Valid	Test	Valid	Test	Valid	Test
$\frac{1}{4}$ grid	MNIST reduced	100 %	99.88 %	97.56 %	99.40 %	99.51 %	97.76 %
	MNIST mixed	100 %	89.87 %	97.76 %	88.90 %	99.27 %	95.63 %
75 superpixels	MNIST reduced			94.13 %	92.70 %	93.64 %	91.62 %
	MNIST mixed			94.13 %	92.90 %	94.62 %	94.17 %

Table 6: Recognition rates on IAM graph datasets

Dataset	**MoNet**		**GraphSAGE**		**Ours**	
	Valid	Test	Valid	Test	Valid	Test
AIDS	80.00%	79.73%	79.60%	79.40%	96.39%	96.93%
Mutagenicity	68.20%	69.41%	69.60%	67.65%	74.00%	76.42%
Web	26.92%	29.74%	40.00%	47.82%	31.79%	31.66%

obtained is that our graph convolution filters are non-oriented because edge attributes are ignored. In the contrary, MoNet loss seems lower on irregular grids. This is possibly due to the fact that the model is less fitted, therefore less prone to overfitting when used on new data. Also, the RAG representation is likely to influence the results. A particular concern on graph convolution operators is sensitivity to domain changes, i.e. capacity to identify similarities on irregular graphs (Experiment 5). Both graph convolution tested show little performance loss between regular (grids) and irregular (75 superpixels RAG) results.

Experiment 5: Visualizing graph convolution on images As an additional experimental material, we tried to visualize the result of a handcrafted filter on images. As for euclidean convolution, the most straightforward filter operation is edge detection. This is usually done by using Sobel operator that calculates intensity gradient at each spatial point of the image.

A potential equivalent graph convolution filter is $(-1\ \ 1)$ (the filter is a 2-nodes graph with respective attributes -1 and 1.) The intuition behind this filter is that the nodes will be matched respectively to the lowest (for the attributed -1 node) and highest (for the attributed 1 node) intensities. As a consequence, this filter will find the highest node attribute difference in every node neighbourhood, making it a sort of eager edge detection filter.

We applied this filter on grid graphs to visualize the output graph as an image (as the graph-to-image transformation is trivial). Figure 3 shows example

Table 7: Recognition rates for different neighbourhood sizes on MNIST reduced 2 class

Representation	1 hop		2 hops	
	Valid	Test	Valid	Test
$\frac{1}{4}$ grid	99.02%	97.55%	98.04%	96.47%
75 superpixels	97.55%	93.74%	96.82%	93.62%

Table 8: Epoch durations on MNIST 2class (Models use different implementations/hardware: CNN is Keras on GPU, MoNet is Theano on GPU and Ours is Keras on CPU)

Representation	CNN	MoNet	Ours
$\frac{1}{4}$ grid	1s	1s	17min 29s
75 superpixels	NA	1s	2min 42s

applications of this filter on both original and rotated examples. This last figure suggests rotation invariance.

Figure 3: MNIST graph convolution examples (respectively original, convoluted and rotated convoluted versions)

Training duration As mentioned in Subsection 3.8, complexity of the model makes experiment tedious to lead. Epoch durations are given in Table 8.

5 Conclusion and perspectives

In this paper, a graph convolutional neural network layer is proposed where graph matching is used as a convolution operator. The proposal was tested in a simplified form. Our model performance is at state of the art level on simple tasks. It shows robustness with respect to graph domain changes. Following improvements could highly benefit to performances and computational costs.

The bipartite solver is not the most suitable choice for our use. Using a less complex solver would allow the full model to be used in practice and applied to larger graphs. More and more combinatorial components are embedded into deep learning architectures [26, 6]. The goal is to make them more efficient but it comes at the price of a higher time complexity. Graph matching has never been used as a convolution operator before and our implementation depends on the graph matching solver. Fast and diffrentiable graph matching solvers like in [26] could be seen as a mean to speed up our model. These solvers rely on full GPU implementations and therefore run in a highly parallel and optimized way. Using the edge information would probably enhance performances significantly as it will probably help with solving more complex problems. Another point

of improvement is regarding differentiation: the gradient must then be approximated by neglecting contribution of the non-differentiable solver intermediary states. Finding a differentiable solver would enhance trainability of the model. Addressing these issues will not only enhance the current degraded version of the model but also allow to implement the full model in a usable form. This model has the peculiarity to learn edge attributes as well as vertex attributes. It is to our knowledge the only graph convolution formulation that suggests to modify the spatiality of edge attributes. Finally, investigating our downsampling layer would justify a whole study for itself. It would be interesting to study the quality of the downsampled graphs but also to study the effect of weighting edges regarding vertex similarity.

Our core contribution is about bringing this new direction that is graph-matching based GCNN and using combinatorial methods in deep learning. This is, from our point of view, the reason why these models are not outperforming the state of the art yet. The work that is presented here is still at its preliminary stage. In another point-of-view, adding complexity to models is also done in hope of increasing learning performances, as for example work involving Transformers [6].

Acknowledgments

This work was supported by a research grant from the Région Centre-Val de Loire, France.

References

[1] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE transactions on pattern analysis and machine intelligence*, 34(11):2274–2282, 2012.

[2] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. F. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gülçehre, H. F. Song, A. J. Ballard, J. Gilmer, G. E. Dahl, A. Vaswani, K. R. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu. Relational inductive biases, deep learning, and graph networks. abs/1806.01261.

[3] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):10008, Oct 2008.

[4] S. Bougleux, L. Brun, V. Carletti, P. Foggia, B. Gaüzère, and M. Vento. Graph edit distance as a quadratic assignment problem. *Pattern Recognition Letters*, 87:38–46, 2017.

[5] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: going beyond euclidean data. *CoRR*, abs/1611.08097, 2016.

[6] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko. End-to-end object detection with transformers. *arXiv preprint arXiv:2005.12872*, 2020.

[7] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *CoRR*, abs/1606.09375, 2016.

[8] I. S. Dhillon, Y. Guan, and B. Kulis. Weighted graph cuts without eigenvectors a multilevel approach. *IEEE transactions on pattern analysis and machine intelligence*, 29(11):1944–1957, 2007.

[9] B. Gaüzere, L. Brun, and D. Villemin. Two new graphs kernels in chemoinformatics. *Pattern Recogn. Lett.*, 33(15):2038 – 2047, 2012.

[10] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. 30, 2017.

[11] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*.

[12] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the 24th international conference on Machine learning*, pages 473–480. ACM, 2007.

[13] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[14] M. Leordeanu, M. Hebert , and R. Sukthankar. An integer projected fixed point method for graph matching and map inference. In *Proceedings Neural Information Processing Systems*, pages 1114–1122, 2009.

[15] Z. Liu and H. Qiao. GNCCP - graduated nonconvexityand concavity procedure. *IEEE Trans. Pattern Anal. Mach. Intell.*, 36:1258–1267, 2014.

[16] F. Monti, D. Boscaini, J. Masci, E. Rodolà, J. Svoboda, and M. M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. *CoRR*, abs/1611.08402, 2016.

[17] A. Nowak, S. Villar, A. S. Bandeira, and J. Bruna. A note on learning algorithms for quadratic assignment with graph neural networks. *CoRR*, abs/1706.07450, 2017.

[18] R. Raveaux, J.-C. Burie, and J.-M. Ogier. Structured representations in a content based image retrieval context. *J. Visual Communication and Image Representation*, 24(8):1252–1268, 2013.

[19] K. Riesen. *Structural Pattern Recognition with Graph Edit Distance - Approximation Algorithms and Applications*. Advances in Computer Vision and Pattern Recognition. Springer, 2015.

[20] K. Riesen and H. Bunke. Iam graph database repository for graph based pattern recognition and machine learning. In N. da Vitoria Lobo, T. Kasparis, F. Roli, J. T. Kwok, M. Georgiopoulos, G. C. Anagnostopoulos, and M. Loog, editors, *Structural, Syntactic, and Statistical Pattern Recognition*, pages 287–297, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[21] K. Riesen and H. Bunke. Approximate graph edit distance computation by means of bipartite graph matching. *Image Vision Comput.*, 27(7):950–959, 2009.

[22] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Trans. Neural Networks*, 20(1):61–80, 2009.

[23] F. Serratosa. Speeding up fast bipartite graph matching through a new cost matrix. *International Journal of Pattern Recognition and Artificial Intelligence*, 29(02):1550010, 2015.

[24] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph Attention Networks. *arXiv e-prints*, page arXiv:1710.10903, Oct 2017.

[25] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. A comprehensive survey on graph neural networks. *CoRR*, abs/1901.00596, 2019.

[26] A. Zanfir and C. Sminchisescu. Deep learning of graph matching. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2684–2693, 2018.

[27] Z. Zhang, P. Cui, and W. Zhu. Deep learning on graphs: A survey. *CoRR*, abs/1812.04202, 2018.

[28] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, and M. Sun. Graph neural networks: A review of methods and applications. *CoRR*, abs/1812.08434, 2018.