Learning Graph Edit Distance by Graph Neural Networks

Pau Riba[a,*], Andreas Fischer[b,c], Josep Lladós[a], Alicia Fornés[a]

[a]*Computer Vision Center, Universitat Autònoma de Barcelona, Spain*
[b]*Department of Informatics, DIVA Group, University of Fribourg, Fribourg, Switzerland*
[c]*Institute of Complex Systems, University of Applied Sciences and Arts Western Switzerland, Fribourg, Switzerland*

Abstract

The emergence of geometric deep learning as a novel framework to deal with graph-based representations has faded away traditional approaches in favor of completely new methodologies. In this paper, we propose a new framework able to combine the advances on deep metric learning with traditional approximations of the graph edit distance. Hence, we propose an efficient graph distance based on the novel field of geometric deep learning. Our method employs a message passing neural network to capture the graph structure, and thus, leveraging this information for its use on a distance computation. The performance of the proposed graph distance is validated on two different scenarios. On the one hand, in a graph retrieval of handwritten words *i.e.* keyword spotting, showing its superior performance when compared with (approximate) graph edit distance benchmarks. On the other hand, demonstrating competitive results for graph similarity learning when compared with the current state-of-the-art on a recent benchmark dataset.

Keywords: Graph Neural Networks, Graph Edit Distance, Geometric Deep Learning, Keyword Spotting, Document Image Analysis.

*Corresponding author
Email addresses: priba@cvc.uab.cat (Pau Riba), andreas.fischer@unifr.ch (Andreas Fischer), josep@cvc.uab.cat (Josep Lladós), afornes@cvc.uab.cat (Alicia Fornés)

1. Introduction

Graph-based representation have been widely used in several application domains such as computer vision [1], bioinformatics [2] or computer graphics [3]. Graphs are powerful and flexible representations able to describe shapes, images, knowledge, etc. in terms of relationships between constituent parts or primitives. In the core of any pattern recognition application, there is the ability to compare two objects. This operation, which is trivial when considering feature vectors defined in $\mathbb{R}^n$, is not properly defined in the graph domain [4, 5]. Due to the inherent graph flexibility, it forces us to adopt some definitions of dissimilarity (similarity) ad hoc to particular purposes. Borgwardt [6] formally defines such problem as follows:

Definition 1.1 (Graph Comparison Problem)**.** Let $g_1 = (V_1, E_1, \mu_1, \nu_1)$ and $g_2 = (V_2, E_2, \mu_2, \nu_2)$ be two graphs from $\mathcal{G}$, the graph comparison problem is to find a function

$$d\colon \mathcal{G} \times \mathcal{G} \to \mathbb{R}$$

such that $d(g_1, g_2)$ quantifies the dissimilarity (similarity) of g_1 and g_2.

Lots of efforts have been made in this direction. In the literature, error-tolerant or inexact graph matching algorithms have been proposed. For instance, Zhou and de la Torre [7] proposed to factorize the large pairwise affinity matrix into smaller matrices that encode, on the one hand, the local node structure of each graph, and on the other hand, the pairwise affinity between both nodes and edges. Moreover, graph embeddings and kernels have been also proposed as mechanisms to compare two graphs. Graph embedding refers to those techniques that aim to explicitly map graphs to vector spaces [8, 9]. Similarly, implicit graph embedding or graph kernel aims at finding a function able to map the input graph into a *Hilbert space* which basically defines a way to compute the similarity between two graphs in terms of a dot product [6, 10]. One of the most popular error-tolerant graph matching methods is the graph edit distance (GED) [11, 12, 13]. Now, the graph comparison problem is formulated in terms

of finding the minimum transformation cost in such a way that an isomorphism exists between the transformed graph g_1 and the second one g_2. In addition, GED algorithms, unlike most embedding and kernel methods, are able to cope with any type of labeled graph and any type of labels on nodes and edges.

The main drawback of GED techniques is that the time complexity is exponential in terms of the number of nodes of the input graphs. Hence, GED is unfeasible in a real scenario, where there may be no constraints in terms of the graph size. Therefore, several algorithms have been proposed to cope with this complexity [14, 15]. However, these approximate algorithms only consider very local node structures in their computation and they do not adapt their costs according to the problem being addressed.

In Euclidean domains such as vectors, images or sequences, deep learning has been proposed as a solution to perform a huge variety of tasks. In the last decade, the particular case of deep neural networks have supposed a breakthrough in computer vision, pattern recognition and artificial intelligence [16]. However, until recent years, deep learning advances were not able to process non-Euclidean data in its framework. Lately, *geometric deep learning*[1] has emerged as a generalization of deep learning methods to non-Euclidean domains such as manifolds and graphs [17]. This field has arisen in the recent years allowing the developed models to encode structural and relational data. Several fields have benefited from this new paradigm, for instance computer vision [1], quantum chemistry [2] and computer graphics [3] among others.

Inspired by the efficient GED approximations and the powerful framework provided by the new advances in geometric deep learning, we propose to leverage its effectiveness as a learning framework to enhance a graph distance computation. Therefore, we are facing a graph metric learning problem. It can be formulated as a contrastive learning problem that finds contrast between similar and dissimilar objects. A siamese architecture is suitable for this problem. Bromley *et al.* [18] proposed it for signature verification. Siamese networks make

[1] `http://geometricdeeplearning.com/`

use of the same model and weights on two separated branches in order to learn a representation where distances can be computed. Later, several approaches have extended this idea, being the triplet loss [19] one of the most successful methods. Recently, novel approaches have focused on extending this concept in order to exploit groups of samples instead of pairs or triplets [20]. Moreover, contrastive learning has been used, not only as a metric learning framework but it has also raised some attention due to its astonishing improvement on unsupervised learning tasks [21].

In this work, we propose to define a triplet learning framework for the graph metric learning problem. In our proposed approach an enriched graph representation is learned by means of a graph neural network. In addition, our proposed distance is based on the Hausdorff Edit Distance introduced by Fischer *et al.* [15] as an efficient approximation of the real graph edit distance. In comparison, our framework has the ability to enrich the initial graph representation by means of message passing operations which learns the edit cost operations. Furthermore, insertion and deletion costs, are dynamically learned according to nodes local contexts. Therefore, we avoid a costly manual process on setting the edit cost operations per each specific problem.

The proposed approach is validated using standard graph datasets for keyword spotting and object classification. In this application scenario, the proposed approach based on a message passing neural network shows competitive results demonstrating the efficacy of our learning framework.

This article supposes a significantly extended version of our previous conference paper [22]. To the best of our knowledge, it was the first work that introduced the idea of learning a graph metric by means of message passing architectures. In this work, we have enhanced our previous graph neural network architecture. Specifically, the main changes from our previous work are (i) a new graph neural network architecture to obtain a better node context representation; (ii) a novel graph similarity which makes use of learned insertions, deletions and substitutions as edit operations; (iii) a learning approach making use of triplets of graphs with in-triplet hard negative mining. Finally, a thor-

ough analysis and evaluation of the involved parameters as well as a performance comparison with the recent state-of-the-art literature is presented.

The rest of this paper is organized as follows. Section 2 introduces the related work on graph neural networks. Section 3 introduce the graph edit distance algorithm along with relevant approximations. Section 4 and 5 proposes our learning framework and learning strategy. Section 6 evaluates the proposed framework. Finally, Section 7 draws the conclusions and future work.

2. Related Work on Geometric Deep Learning

In the following, some basic notations, definitions and previous works are presented in the context of geometric deep learning. As mentioned in the introduction, *Geometric deep learning* has emerged as a generalization of deep learning methods to non-Euclidean domains such as graphs and manifolds [17]. In this document we will specifically focus on its applications to graphs.

A graph can be roughly defined as a symbolic data structure describing relations (*edges*) between a finite set of objects (*nodes*). Graphs are formally defined as

Definition 2.1 (Graph)**.** Let L_V and L_E be a finite or infinite label sets for nodes and edges, respectively. A *graph* g is a $4-$tuple $g = (V, E, \mu, \nu)$ where,

- V is the finite set of *nodes*, also known as *vertices*,
- $E \subseteq V \times V$ is the set of *edges*,
- $\mu \colon V \to L_V$ is the node labelling function, and
- $\nu \colon E \to L_E$ is the edge labelling function.

We denote $|V|$ and $|E|$ as the *order* and *size* of a graph, namely, the number of nodes and edges respectively. Moreover, the *neighborhood* of a given node $v \in V$ in a graph g is defined as the set of nodes $\{v_i\}$ adjacent to v. We denote the neighborhood as $\mathcal{N}(v)$.

2.1. Graph neural networks

Graph Neural Networks were first proposed by Gori and Scarselli [23, 24] as the first attempts to generalize neural networks to graphs. Later, Bruna *et al.* [25] proposed the first formulation of CNNs on graphs taking advantage of the graph spectral domain. However, the ability to process graph data came with a huge time complexity becoming inappropriate for real scenarios. Later, the works of Henaff *et al.* [26], Defferrard *et al.* [27] and Kipf *et al.* [28] addressed these computational drawbacks. In its simplest form, a GNN layer is defined as

$$h^{(k+1)} = G_c(h^{(k)}) = \rho\left(\sum_{B\in\mathcal{A}^{(k)}} Bh^{(k)}\Theta_B^{(k)}\right), \tag{1}$$

where $h^{(k)}$ is the node hidden state at the k-th layer, ρ is a non-liniarity such as $\mathrm{ReLU}(\cdot)$, $\mathcal{A}$ is a set of graph intrinsic linear operators that act locally on the graph signal and Θ are learnable parameters. The set of graph intrinsic linear operators can handle multi-relational graphs, however, in most cases, $\mathcal{A}$ only contains the adjacency matrix.

Recently, Gilmer *et al.* [2] proposed an approach named *Message Passing Neural Networks* (MPNNs) as a general supervised learning framework for graphs. This approach is able to generalize several GNN layers to a common pipeline. They propose to define each layer by means of two differentiable functions, on the one hand, a message function $M^{(k)}(\cdot)$ which collects the information from the neighboring nodes and edges according to

$$m_v^{(k+1)} = \sum_{u\in\mathcal{N}(v)} M^{(k+1)}(h_v^{(k)}, h_u^{(k)}, e_{vu}), \tag{2}$$

where $h_u^{(k)}$ and $h_v^{(k)}$ are the hidden states of nodes v and u at iteration k and $\mathcal{N}(v)$ denotes the neighbours of v in the graph g. On the other hand, an update function which updates the hidden state of the central node v according to the

message $m_v^{(k+1)}$. The update function is formally defined as

$$h_v^{(k+1)} = U^{(k+1)}(h_v^{(k)}, m_v^{(k+1)}). \tag{3}$$

Several important GNN layers have been proposed along the last years, the most relevants for the scope of this work are the Graph Attention Networks (GAT) [29] and the Gated Graph Neural Networks (GG-NN) [30].

The literature on graph neural networks and their applications is quite large, so we refer the interested readers to recent survey papers for a comprehensive overview of these methodologies [17, 31, 32, 33]. Very recently, Dwivedi *et al.* [34] presented a reproducible benchmarking framework.

2.2. Graph metric learning

Neural networks have been widely used as the learning framework for similarity problems. Promptly, siamese neural networks were adopted as a family of neural networks consisting of two networks with shared weights for similarity learning. For example, Baldi *et al.* [35] makes use of a siamese model for fingerprint recognition whereas Bromley *et al.* [18] presented a siamese architecture for signature verification. Siamese neural networks use a pair of samples to train with positive and negative examples *i.e.* being similar or not. Later, several approaches have extended this idea in order to take always into account positives and a negatives examples. These approaches are known as triplet networks [19]. In this case, three networks with shared weights are used to bring similar examples together and dissimilar examples to be far apart.

Inspired on these works, several papers appeared extending these ideas to the graph domain. Thus, we proceed to review a handful of approaches facing the graph similarity learning problem. Li *et al.* [36] presented two different models to solve the graph similarity problem, on the one hand a graph embedding model, and on the other hand, a graph matching network. Both models can be trained with pairs or triplets. Let us briefly review each one of these models,

- **Graph embedding model**: This model, takes advantage of siamese

GNN's to embed the given graphs into a vectorial space. Then, given the pair of vectorial representations, a similarity metric in the vector space can be computed by means of the Euclidean, cosine or Hamming similarities. Very similar approaches have also been presented in other works, for instance Chaudhuri *et al.* [37] who trained a similar approach with contrastive loss or the work introduced by Zhang *et al.* [38] which uses the L_2 loss to mimic the real similarity score.

- **Graph matching networks (GMN)**: Similarly to the previous architecture, two GNNs with shared weights process the input graphs. However, in this case, the authors propose to modify the node update module in order to take into account not only the aggregated messages on the edges of each graph, but a cross-graph message which measures how well the nodes match from one graph to the other. Finally, following the same idea as the previous model, each graph is finally converted into a vectorial representation which is later used in a similarity metric.

Another interesting approach, namely SimGNN, was proposed by Bai *et al.* [39]. In this work, the authors proposed to combine graph-level embeddings and node-node similarity scores by taking their histogram of features. However, as the histogram function is not differentiable, this methodology still relies on the graph-level embedding for computing the final similarity score. Their model is trained according to the real graph edit distance for small datasets whereas the smallest distance computed by three well-known approximate algorithms is taken to handle large datasets. The authors extended this work by proposing a new model named GraphSim [40]. In this architecture, only three node-node similarities scores are used corresponding to node embeddings at different scales. After that, the similarity matrices are treated as images and a CNN is used to process them to discover the optimal node matching pattern. However, to deal with the permutation invariant ordering of graph nodes, they propose a BFS ordering. It also allows the use of CNN as they claim that the required spatial locality performs properly. Moreover, the similarity matrices are first padded

to $\max(|V_1|, |V_2|)$, where V_1 and V_2 are the node sets of the graphs involved and resized to meet the expected size. Finally, following the SimGNN training, a precomputed similarity score is used to lead the training.

Compared to these works, our model makes use of a node-node distance matrix to obtain a global graph distance metric. Therefore, we are not obtaining a global vectorial representation of our graphs nor applying cross-convolution layers in our graph neural network architecture. This allows us to make use of any differentiable graph or set distance, which preserves the permutation invariance property, while avoiding the computational overhead of the cross-convolution layers. Moreover, we avoid the loss of structural information of other approaches when obtaining a vectorial graph representation by explicitly dealing with the structure in the distance itself.

3. Related work on Graph Edit Distance

This Section introduces the concept of graph edit distance (GED) jointly with relevant cubic and quadratic time approximations.

3.1. Definition

The *Graph Edit Distance* (GED) [13, 41, 12] evaluates the similarity of two graphs in terms of edit operations. The GED is inspired by the *String Edit Distance* (SED), also known as the *Levenshtein* distance [42, 43]. Actually, it can be seen as a generalisation of SED as strings can be viewed as a special case of graphs. In this specific case, the order of the characters allows the efficient use of dynamic programming to find the string-to-string correspondence. However, for general graphs, the correspondence cannot rely on a specific ordering of nodes and edges.

The main idea of GED is to compute the minimum cost transformation from the source graph g_1 to the target one g_2 in terms of a sequence of edit operations $e_1, \ldots, e_k$. This sequence of edit operations is named *edit path* between g_1 and g_2. GED is formally defined as

Definition 3.1 (Graph Edit Distance)**.** Let $g_1 = (V_1, E_1, \mu_1, \nu_1)$ and $g_2 = (V_2, E_2, \mu_2, \nu_2)$ be the source and the target graphs respectively. The *graph edit distance* between g_1 and g_2 is defined by

$$d(g_1, g_2) = \min_{(e_1,\ldots,e_2)\in\Upsilon(g_1,g_2)} \sum_{i=1}^{k} c(e_i),$$

where $\Upsilon(g_1, g_2)$ denotes the set of edit paths transforming g_1 into g_2, and $c(e_i)$ denotes the cost function measuring the strength of the edit operation e_i.

Usually the considered edit operations are *insertion*, *deletion* and *substitutions* for nodes and edges. For instance, a frequently used cost functions for nodes and edges with labels defined in $\mathbb{R}^n$ is

$$\begin{aligned}
&\mathcal{C}(u \longrightarrow v) = \alpha \cdot \| \mu_1(u) - \mu_2(v) \| \\
&\mathcal{C}(u \longrightarrow \varepsilon) = \alpha \cdot \tau_n, \quad \mathcal{C}(\varepsilon \longrightarrow v) = \alpha \cdot \tau_n \\
&\mathcal{C}(p \longrightarrow q) = (1-\alpha) \cdot \| \nu_1(p) - \nu_2(q) \| \\
&\mathcal{C}(p \longrightarrow \varepsilon) = (1-\alpha) \cdot \tau_e, \quad \mathcal{C}(\varepsilon \longrightarrow q) = (1-\alpha) \cdot \tau_e
\end{aligned} \tag{4}$$

for nodes $u \in V_1$, $v \in V_2$ and edges $p \in E_1$, $q \in E_2$ and user-defined parameters $0 \leq \alpha \leq 1$ indicating a trade-off between node and edge costs and $\tau_n, \tau_e > 0$ to fix the deletion and insertion costs for nodes and edges respectively.

The graph edit distance is a known NP-complete problem (see [44] for a detailed proof), exponential with respect to the number of nodes. Thus, in addition to exact GED algorithms, some efficient approximations have been proposed [45, 46]. In the following we will review two techniques that have been widely adopted in the literature.

3.2. Assignment edit distance

The *assignment edit distance* (AED), also known as *bipartite graph matching*, proposed by Riesen *et al.* [14], is a cubic time approximation of GED with

respect to the number of nodes of the involved graphs. It provides an upper bound of order $\mathcal{O}\left((n_1+n_2)^3\right)$ where $n_1 = |V_1|$ and $n_2 = |V_2|$.

The main idea is to transform the GED computation to an assignment problem between nodes and their local structure. This method, defines a matrix of edit costs between the nodes of both graphs. Afterwards, the best correspondence between nodes is found by a linear assignment method [47]. The matrix definition for the AED algorithm takes into consideration both, the local structure of the vertices and their attributes. The cost matrix C is defined as

$$C = \left[\begin{array}{ccc|ccc} c_{1,1} & \cdots & c_{1,m} & c_{1,\varepsilon} & \cdots & \infty \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ c_{n,1} & \cdots & c_{n,m} & \infty & \cdots & c_{n,\varepsilon} \\ \hline c_{\varepsilon,1} & \cdots & \infty & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \infty & \cdots & c_{\varepsilon,n} & 0 & \cdots & 0 \end{array}\right]$$

where $c_{i,j}$ denotes the cost of a node substitution $c(u_i \rightarrow v_j)$; $c_{i,\varepsilon}$ denotes the cost of a node deletion $c(u_i \rightarrow \varepsilon)$; and $c_{\varepsilon,j}$ denotes the costs of a node insertion $c(\varepsilon \rightarrow v_j)$ where $v_i \in V_1$ and $v_j \in V_2$.

Other works have focused on speeding-up this method in some particular settings. For example, Serratosa *et al.* [48] define an algorithm able to reduce the computation time with the only constrain that the edit costs should define the graph edit distance as a real distance function, that is, the cost of insertion plus deletion of nodes and edges have to be lower or equal than the cost of substitution of nodes and edges.

3.3. Hausdorff Edit Distance

Despite obtaining a good approximation, the time complexity of the AED algorithm is still a problem for some applications where the time is an important constrain. In order to alleviate this issue, Fischer *et al.* [15] proposed the *Hausdorff Edit Distance* (HED) which is a lower bound approximation of the

real GED with a quadratic time complexity of $\mathcal{O}(n_1 \cdot n_2)$ where $n_1 = |V_1|$ and $n_2 = |V_2|$. HED is based on the *Hausdorff distance* is formally defined as

Definition 3.2 (Hausdorff distance)**.** Let A and B be two non-empty subsets of a metric space (M, d). The Hausdorff distance $d_H(A, B)$ is defined as

$$d_H(A, B) = \max\left(\sup_{a\in A}\inf_{b\in B} d(a, b), \sup_{b\in B}\inf_{a\in A} d(a, b)\right).$$

For finite sets A,B the Hausdorff distance is reformulated as

$$d_H(A, B) = \max\big(\max_{a\in A}\inf_{b\in B} d(a, b), \max_{b\in B}\inf_{a\in A} d(a, b)\big).$$

By definition, the Hausdorff distance is very sensitive to outliers. Hence, in their work they propose to replace the maximum operator with the summation operation, which forces the distance to take into account all nearest neighbour distances and becomes more robust to noise than the original one. Thus, the new distance is defined as

$$d_{\hat{H}}(A, B) = \sum_{a\in A}\min_{b\in B} d(a, b) + \sum_{b\in B}\min_{a\in A} d(a, b). \tag{5}$$

Until now, this distance $d_{\hat{H}}(\cdot)$ do not consider node insertions and deletions. Therefore, from Equation 5 they define a new distance on graphs. Given two graphs $g_1 = (V_1, E_1, \mu_1, \nu_1)$ and $g_2 = (V_2, E_2, \mu_2, \nu_2)$ and a matching cost defined as c, the HED is defined as,

$$\mathrm{HED}(g_1, g_2, c) = \sum_{u\in V_1}\min_{v\in V_2\cup\{\varepsilon\}} c_n^*(u, v) + \sum_{v\in V2}\min_{u\in V_1\cup\{\varepsilon\}} c_n^*(v, u), \tag{6}$$

where $c_n^*(u, v)$ is a modified node matching cost defined as,

$$c_n^* = \begin{cases} \frac{c_n(u,v)}{2}, & \text{if } (u \to c) \text{ is a substitution} \\ c_n(u, v), & \text{otherwise.} \end{cases}$$

This redefinition of the node matching cost is needed because HED does not enforce bidirectional substitutions. Equation 6 is composed by a summation, hence, only if both directions are considered, the full cost will be taken into account. The same matching algorithm is considered, if needed, for the edge matching. In this setting, the HED finds an optimal assignment per node instead of a global optimal assignment as pretended by the real GED.

A typical drawback of GED approximation algorithm is that it only relies on local edge structures rather than global information. Some efforts have been made to improve the performance by increasing the node context at matching time [49]. However, obtaining a better knowledge on the relation of each node within the graph is still an open issue that we aim to address by the new advances on graph neural networks.

4. The Learned Graph Distance Framework

This section is devoted to present our proposed learning framework for graph distance. The proposed model learns the Hausdorff edit distance, proposed by Fischer *et al.* [15]. As a learning setting, the proposed architecture can be trained either with pairs or triplets of graphs. Hence, as ground-truth, only the information on whether or not two graphs belong to the same class is required. Note, that in our proposed approach, we do not require the node correspondence information nor the real graph edit distance. Instead, the node assignment is implicitly learned by our system. Moreover, the edit costs for both insertions, deletions and substitutions are learned by our framework. Therefore, we do not require to manually tune these parameters following the traditional pipeline. Although our framework can be trained using pairs, in this work we will focus on the triplet setting. Hence, we make use of three GNN with shared weights.

Figure 1 shows a graphical outline of our proposed approach. Our pipeline can be divided in two stages. Firstly, a graph neural network $\phi(\cdot)$ is used to obtain a node-level embedding which codifies the local context information, in terms of structure, for each node. Secondly, a novel graph similarity algorithm

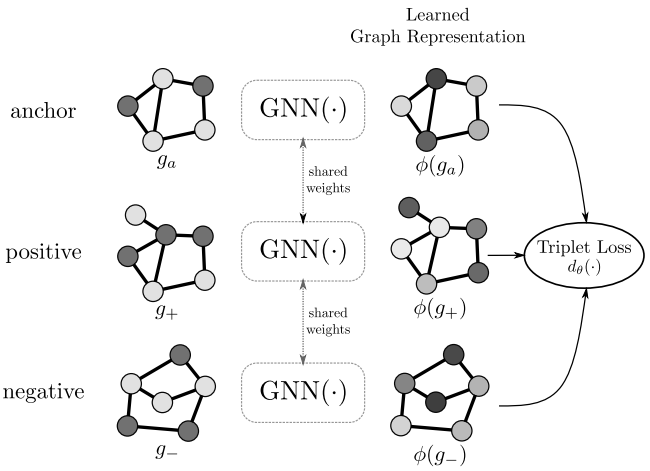


Figure 1: Overview of our learning framework. Given a triplet of graphs (g_a, g_+, g_-) as the anchor, positive and negative samples respectively, the GNN $\phi(\cdot)$ learns a graph representation per each one $(\phi(g_a), \phi(g_+), \phi(g_-))$, which can be matched by means of a learned distance $d_\theta(\cdot)$.

based on the Hausdorff edit distance is proposed as a technique to compare two graphs $d_\theta(\cdot)$. Observe that the graph similarity can be replaced by any differentiable graph distance approach.

Each stage is carefully described next. First, the GNN architecture is explained in detail. Afterwards, the proposed graph similarity is developed.

4.1. Learning node embeddings

The first stage of our framework is a graph neural network architecture $\phi(\cdot)$ able to learn a new graph representation in terms of node embeddings. Our architecture consists of K propagation layers that map the input graph to an enriched representation. Thus, each propagation layer takes a set of node representations at layer k, $\{h_i^{(k)}\}_{i \in V}$ and maps it to a new node representation $\{h_i^{(k+1)}\}_{i \in V}$ at layer $k+1$. We evaluate the following two different architectures according to two different message passing strategies.

GAT-based model: This model uses graph attention networks (GAT), introduced by Velikovi *et al.* [29]. A GAT layer is formally defined as

$$h_v^{(k+1)} = \sum_{u \in \mathcal{N}(v)} \alpha_{vu} W^{(k)} h_u^{(k)}, \tag{7}$$

where α_{vu} is the attention score between node v and node u, $W^{(k)}$ are the learned weights and $h_v^{(k)}$ is the hidden state of node v, both, at layer k. The attention score α_{vu} is learned by

$$\begin{aligned} \alpha_{vu}^{(k)} &= \mathrm{softmax_v}(e_{vu}^{l}) \\ e_{vu}^{(k)} &= \mathrm{LeakyReLU}\left(\vec{a}^{t}[W^{(k)}h_v^{(k)} \| W^{(k)}h_u^{(k)}]\right), \end{aligned} \tag{8}$$

where $\vec{a}$ and $W^{(k)}$ are a vector and a matrix of learned weights. Moreover, a multi-head attention can be used to enrich the model capacity and to stabilize the learning process.

In our setting, we use these layers with residual connections, four attention heads and BatchNorm layers [50] with the exception of the last layer. The attention heads are concatenated at the intermediary layers and averaged for the final layer.

GRU-based model: This architecture is based on the gated graph neural networks (GG-NN) proposed by Li *et al.* [30]. Originally, the message function is formulated as $M(h_v^{(k)}, h_u^{(k)}, e_{vu}) = A_{e_{vu}}h_u^{(k)}$, where $A_{e_{vu}}$ is a learned matrix for each possible edge label. Note that we are restricted to a discrete set of labels. In order to overcome this constrain, Gilmer *et al.* [2] proposed to use a modified message function defined as $M(h_v^{(k)}, h_u^{(k)}, e_{vu}) = A(e_{vu})h_u^{(k)}$, where $A(e_{vu})$ is a neural network which maps the edge vector to a matrix $d \times d$. This modification allows the use of non-discrete information as edge attributes. Finally, the update function is defined as $U(h_v^{(k)}, m_v^{(k)}) = \mathrm{GRU}(h_v^{(k)}, m_v^{(k)})$, where GRU is the Gated Recurrent Unit [51]. In its original formulation, the first node hidden state is padded with zeros to meet the size defined by the GRU, however, we propose to use a fully-connected layer as a first node embedding. Moreover, we propose to incorporate edge features according to the source and destination node according to, $e_{vu} = \mathrm{MLP}(|h_v^{(1)} - h_u^{(1)}|)$ as proposed in [52]. There, MLP stands for multi-layer perceptron and the absolute value is used to preserve the simmetry of the edge direction.

4.2. Graph Distance or Similarity

Following the idea of HED defined in Equation 6, we propose to dynamically adapt the insertions and deletion costs according to the application domain. With this aim, we introduce two learnable costs $c(\varepsilon \to v)$ and $c(u \to \varepsilon)$ for the insertion and deletion of nodes. Thus, taking advantage of the computed node embeddings, our nodes are enriched with information aggregated from their local context and, therefore, its importance within the graph. Thus, we propose to take advantage of this in order to define two neural networks $\varphi_i(\cdot)$ and $\varphi_d(\cdot)$, defined as $\varphi_* \colon \mathbb{R}^n \to \mathbb{R}^+$, able to decide the corresponding cost of this operation. In our experiments, $\varphi_i(\cdot)$ and $\varphi_d(\cdot)$ are the same network with shared weights as we consider the insertion and deletion operations to be symmetric. Moreover, we take the absolute value as the insertion and deletion costs must be positive.

Therefore, we define the distance between two graphs $g_1 = (V_1, E_1, \mu_1, \nu_1)$ and $g_2 = (V_2, E_2, \mu_2, \nu_2)$ as

$$d_\theta(g_1, g_2) = \frac{1}{|V_1| + |V_2|} \left(\sum_{u \in V_1 \cup \{\varepsilon\}} \min_{v \in V_2 \cup \{\varepsilon\}} c_\theta(u, v) + \sum_{v \in V_2 \cup \{\varepsilon\}} \min_{u \in V_1 \cup \{\varepsilon\}} c_\theta(u, v) \right), \tag{9}$$

where θ are learnable parameters and $c_\theta(\cdot, \cdot)$ is the corresponding learnable cost function defined as

$$c_\theta(u, v) = \begin{cases} \varphi_d(u; \theta) & \text{if } (u \to \varepsilon) \text{ is a deletion,} \\ \varphi_i(v; \theta) & \text{if } (\varepsilon \to v) \text{ is an insertion,} \\ \frac{d(u,v)}{2} & \text{otherwise.} \end{cases} \tag{10}$$

In our scenario, the edges are not taken into account as we consider the local structures to be already encoded during the message passing phase. However, edges can be incorporated to Equation 10, considering the adjacent edges as nodes and applying the same distance $d_\theta(\cdot)$ with different learned weights.

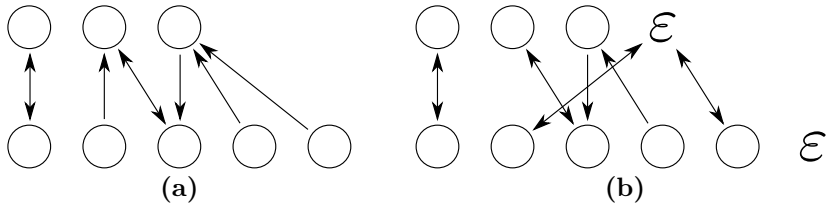


Figure 2: Assignment problem according to the proposed distance. (a) only substitutions are considered, (b) insertions and deletions are included as an extra epsilon node.

Observe that an important aspect of the proposed distance is the fact that the node correspondence might not be symmetric. Figure 2 illustrates this issue, moreover, we also show the effect of considering insertions and deletions as a ε node in Figure 2(b). Note that not considering ε nodes as proposed in [22] and illustrated in Figure 2(a), can limit the learning capabilities of the framework.

A limitation of our approach is that in some scenarios it might lose some node feature information in favor of encoding the local node structure. As a solution, we optionally combine the original graph information into Equation 10, that can be redefined as,

$$c_\theta(u,v) = \begin{cases} \tau_d + \varphi_d(u;\theta) & \text{if } (u \to \varepsilon) \text{ is a deletion,} \\ \tau_i + \varphi_i(v;\theta) & \text{if } (\varepsilon \to v) \text{ is an insertion,} \\ \frac{d(u,v)+d'(u,v)}{2} & \text{otherwise,} \end{cases} \tag{11}$$

where $\tau_d, \tau_i > 0$ are user-defined parameters to fix the minimum cost for node deletion and insertion respectively; $d'(\cdot,\cdot)$ corresponds to the distance computed on the original node attributes. We find this setting helpful to avoid incorrect matchings between nodes that share structurally similar neighborhoods.

5. Training setting and learning objective

In this work, we follow the idea of triplet networks to exploit the ranking properties of the desired metric. Thus, we use three GNN models with shared weights following the architecture illustrated in Figure 1.

Our model is trained in a supervised manner, so we know which pairs of graphs belong to the same class. Compared to other approaches, we do not require node assignments nor a pre-computed similarity score. All models were trained using the *Adam* optimizer [53] with weight decay *i.e.* L_2 regularization. The learning rate of 0.001 is multiplied by 0.95 every 5 epochs to decrease its value, and we applied early stopping to finish our training process.

The objective function to minimize is the triplet loss, also known as margin ranking loss. This learning objective receives three samples in which we already know its ranking, *i.e.* which pair should have a higher similarity score or distance. Let $\{g_a, g_+, g_-\}$ be a triplet training sample where, g_a is the anchor graph, g_+ is a positive graph sample *i.e.* a sample different from g_a but belonging to the same class and g_- is a negative graph example *i.e.* a sample belonging to a different class. Then, the triplet loss is defined as

$$\mathcal{L}(\delta_+, \delta_-) = \max(0, \mu + \delta_+ - \delta_-), \tag{12}$$

where μ is a fixed margin parameter, $\delta_+ = d_\theta(\phi(g_a), \phi(g_+))$ is the distance with respect to the positive sample and $\delta_- = d_\theta(\phi(g_a), \phi(g_-))$ is the distance with respect to the negative sample. Figure 3 illustrates how this loss performs. Note that positive pairs are pushed to be close each other whereas negative samples are separated at least by the predefined margin μ.

Moreover, following the idea introduced in [54], we apply an in-triplet hard negative mining which means that the anchor and positive samples can be swapped in case the positive sample is harder than the anchor one. Hence, we define $\delta'_- = d_\theta(\phi(a), \phi(n))$ and $\delta_* = \min(\delta_-, \delta'_-)$. Finally, the new loss with the anchor swap is defined as

$$\mathcal{L}(\delta_+, \delta_*) = \max(0, \mu + \delta_+ - \delta_*). \tag{13}$$

Algorithm 1 presents our training strategy. $\Gamma(\cdot)$ denotes the optimizer function.

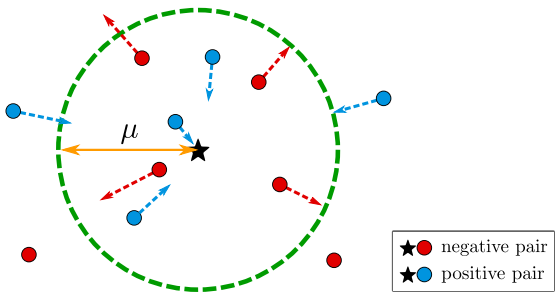


Figure 3: Illustration of the triplet learning objective. The anchor graph illustrated as a star should be close to its positive pair, in blue, and farther than a margin μ to its negative counterpart.

Algorithm 1 Training algorithm for our proposed model.

Require: Input data $\mathcal{G}$; max training iterations T
Ensure: Networks parameters $\Theta = \{\Theta_\phi, \theta\}$.

1: **repeat**
2: Sample triplet mini-batches $\{g_a, g_+, g_-\}_{i=1}^{N_B}$
3: $\mathcal{L} \leftarrow$ Eq. 13
4: $\Theta \leftarrow \Theta - \Gamma(\nabla_\Theta \mathcal{L})$
5: **until** Convergence or max training iterations T

6. Experimental Validation

For validating our approach, a keyword spotting application in historical manuscripts has been considered as our main application scenario. Moreover, a final experiment on a classical mesh graph dataset is conducted. Our empirical evaluation demonstrate that the proposed approach provides competitive results when compared to the state-of-the-art. All the code is available at `github.com/priba/graph_metric.pytorch`.

6.1. Historical Keyword Spotting

In Document Image Analysis and recognition, Keyword Spotting (KWS), also known as word spotting, has emerged as an alternative to handwritten text recognition for documents in which the transcription performance is not satisfactory. Therefore, KWS is formulated as a content-based image retrieval strategy which relies upon obtaining a robust word image representation and a subsequent retrieval scheme.

6.1.1. Dataset Description

The HistoGraph dataset [55, 56] is a graph database for historical keyword spotting evaluation[2]. It consists of different well known manuscripts.

George Washington (GW) [57]: This database is based on handwritten letters written in English by George Washington and his associates during the American Revolutionary War in 1755[3]. It consists of 20 pages with a total of $4,894$ handwritten words. Even tough several writers were involved, it presents small variations in style and only minor signs of degradation.

Parzival (PAR) [57]: This collection consists of 45 handwritten pages written by the German poet Wolfgang Von Eschenbach in the 13th century. The manuscript is written in Middle High German with a total of $23,478$ handwritten words. Similarly to GW, the variations caused by the writing style are low, however, there are remarkable variations caused by degradation.

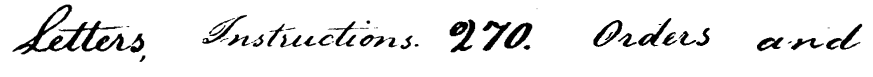
Alvermann Konzilsprotokolle (AK) [58]: It consists of German handwritten minutes of formal meetings held by the central administration of the University of Greifswald in the period of 1794 to 1797. In total $18,000$ pages were used with small variations in style and only minor signs of degradation.

Botany (BOT) [58]: It consists of more than 100 different botanical records made by the government in British India during the period of 1800 to 1850. The records are written in English and contain certain signs of degradation and especially fading. The variations in the writing style are noticeable especially with respect to scaling and intra-word variations.

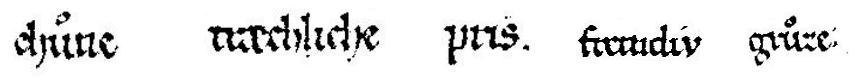
Figure 4 provides some examples of pre-processed word images from which the graphs are created. Observe that the word segmentation of AK and BOT datasets is imperfect [58]. Moreover, these two datasets do not provide a validation set. So, some images from the training set have been used for validation. Table 1 provides an overview of the dataset in terms of number of words.

[2]Available at `http://www.histograph.ch/`

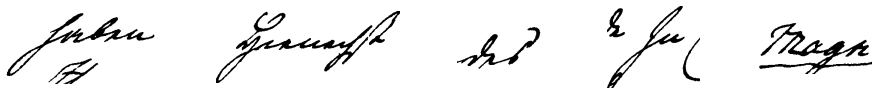
[3]George Washington Papers at the Library of Congress from 1741-1799, Series 2, Letterbook 1, pages 270-279 and 300-309, `https://www.loc.gov/collections/george-washington-papers/about-this-collection/`



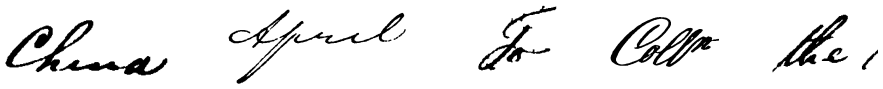
(a) George Washington (GW)



(b) Parzival (PAR)



(c) Alvermann Konzilsprotokolle (AK)



(d) Botany (BOT)

Figure 4: Pre-processed word examples of the four datasets.

Table 1: Dataset overview in terms of number of keywords and word images for training, validating and testing respectively

Dataset	Keywords	Train	Validation	Test
GW	105	2,447	1,224	1,224
PAR	1,217	11,468	4,621	6,869
BOT	150	1,684	-	3,380
AK	200	1,849	-	3,734

To obtain a graph for each word in these datasets, the two most promising graph constructions introduced in [56] have been used:

- **Keypoint**: Characteristic points are extracted from the skeletonized word image. Moreover between the connected characteristic points, equidistant nodes are inserted on top of the word skeleton.
- **Projection**: An adaptative grid is generated according to the vertical and horizontal projection profiles. Then, nodes are inserted in the corresponding center of mass of each grid cell. Moreover, undirected edges are inserted if nodes are directly connected by a stroke.

All the datasets presented in this work only contain spatial and structural

information. This means that nodes are labelled with its normalized (x,y)-position on the image and that edges are unlabelled.

6.1.2. Experimental protocol

The experiments reported in this section use $K = 3$ GNN layers and, for the edit cost operation described in Equation 11, we experimentally set the parameters τ_i and τ_d to 0.5. Following the evaluation schemes of previous word spotting methodologies, we performed our experiments on two evaluation protocols.

- **Individual**: Each query image follows the traditional retrieval pipeline. Thus, queries are matched against the elements in the gallery and each ranking is evaluated independently.
- **Combined**: A query can consist of a set of graphs $Q = \{q_1, \ldots, q_t\}$ where all the instances $q \in Q$ represent the same keyword. In this case, we consider the minimal distance achieved on all t query graphs. This second evaluation protocol was adopted in some previous graph-based word spotting works [59, 56]. The motivation of this setting is to mitigate the structural bias provided by the query instance, *i.e.* different handwriting styles can provide extremely different graphs, so, in terms of graph distances, it is unrealistic to consider them from the same class.

For the evaluation, we use the mean Average Precision (mAP), a classic information retrieval metric [60]. First, let us define Average Precision (AP) as

$$\mathrm{AP} = \frac{\sum_{n=1}^{|\,\mathrm{ret}\,|} P@n \times r(n)}{|\,\mathrm{rel}\,|}, \tag{14}$$

where $P@n$ is the precision at n and $r(n)$ is a binary function on the relevance of the n-th item in the returned ranked list. Then, the mAP is defined as:

$$\mathrm{mAP} = \frac{\sum_{q=1}^{Q} \mathrm{AP}(q)}{Q}, \tag{15}$$

where Q is the number of queries.

6.1.3. Ablation study

We first empirically investigate the influence of the margin parameter μ to each model, as well as the importance of the GNN layers choice. Table 2 presents a comparison of the different settings, providing the averaged mAP of 4 runs and its corresponding standard deviation. This evaluation has been done for all the datasets and for both graph representations *viz.* Keypoint and Projection. The evaluation protocol in this experiment is Individual as we believe that it is the natural experimental setting for this problems.

From these results, we observe that GRU-based models are slightly better, allowing a higher degree of deformations between words from the same class. Note that both datasets AK and BOT do contain imperfect word segmentations. In addition, these datasets contain samples written with a more artistic calligraphic style as shown in Figure 4. Thus, the artistic strokes are drivers of a higher degree of complexity. Observe that the performance drop in BOT dataset can be also explained by the artistic nature of the dataset.

Additionally, the Keypoint representation performs the best on the GW dataset, since the strokes are simpler and its structure in terms of the binary image skeleton is more relevant for a proper retrieval. Finally, we observe that a higher margin μ is more adequate for the GAT-based models whereas it's harmful for the GRU-based one.

6.1.4. Results and discussion

Table 3 compares with graph-based methodologies. In this setting we follow the Combined evaluation protocol reported by [59, 56]. For each dataset and graph representation, we use the best model reported in the previous section. Observe that, for a fair comparison, that is, using the same graph representation, we outperform both AED [14] and HED [15] on all the datasets but AK, where we obtain very similar results. However, the ensemble methods reported in [56] are able to obtain a better performance on the datasets with more variability. Note that these ensembles combine, in different ways, the graph distances computed on different graph representations of the same images. Therefore, we do

Table 2: Study on the GNN model and margin parameter of the proposed model. Mean average precision (mAP) and standard deviation (average on four runs) for graph-based KWS system on George Washington (GW), Parzival (PAR) Alvermann Konzilsprotokolle (AK) and Botany (BOT) datasets.

	Model	μ	GW		PAR		AK		BOT	
			mAP	±	mAP	±	mAP	±	mAP	±
Keypoint	GAT	1	72.49	1.169	66.46	3.162	62.90	1.325	39.86	0.396
		10	**76.92**	2.309	**73.14**	0.973	62.72	1.783	**41.52**	0.782
	GRU	1	72.86	3.331	67.27	1.281	**64.42**	1.003	39.69	0.532
		10	68.45	2.715	48.59	9.571	60.84	1.127	38.22	0.778
Projection	GAT	1	67.86	2.379	70.77	1.906	63.44	1.233	39.12	2.037
		10	**70.25**	3.431	**75.19**	0.755	62.72	1.518	38.83	2.801
	GRU	1	68.09	1.234	71.07	1.933	**65.04**	1.226	**42.83**	0.568
		10	63.39	4.222	52.32	1.298	60.51	1.451	37.59	0.778

not consider it a fair comparison but a remarkable fact that the performance of our system is able to outperform them in two of the datasets while obtaining competitive results on the other two.

Table 3: State-of-the-art on graph-based KWS techniques. Mean average precision (mAP) for graph-based KWS system on GW, PAR, AK and BOT datasets.

Distance	Representation		GW	PAR	AK	BOT
AED [14]	Keypoint [59]		68.42	55.03	77.24	50.94
	Projection [59]		60.83	63.35	76.02	50.49
	Ensemble [56]	min	70.56	67.90	82.75	65.19
		max	62.58	67.57	82.09	67.57
		mean	69.16	79.38	84.25	**68.88**
		sum_α	68.44	74.51	**84.77**	68.77
		sum_{map}	70.20	76.80	84.25	**68.88**
HED [15]	Keypoint [59]		69.28	69.23	79.72	51.74
	Projection [59]		66.71	72.82	**81.06**	51.69
Ours	Keypoint		**78.48**	79.29	78.64	51.90
	Projection		73.03	**79.95**	79.55	**52.83**

Table 4 shows a comparison with non-graph based approaches. In particular, we compare against three state-of-the-art learning-based reference systems of the ICFHR2016 competition [58]. In this case, the evaluation of these learning-based

frameworks follows the protocol described in the competition. Thus, queries of the same query keyword are considered to be independent. Note that, due to this query protocol, the learning-based frameworks are not directly comparable to the state-of-the-art graph-based KWS results that we have reported above. In this table, we can observe the superiority of learning methods working directly on the image domain. In particular, PHOCNet [61] leads to stunning accuracies for this task. However, the proposed graph-based approach, is able to provide a new step towards closing the gap between structural and statistical methodologies on this kind of tasks. In addition, the proposed approach is able to deal with the noise introduced by the graph construction.

Table 4: Comparison against non-graph learning based systems. Mean average precision (mAP) for graph-based KWS system on AK and BOT datasets.

Method	Representation	AK	BOT
CVCDAG [62]	-	77.91	75.77
PHOCNet [61]	-	96.05	89.69
QTOB [63]	-	82.15	54.95
Ours	Keypoint	64.42	41.52
	Projection	65.04	42.83

Finally, Figure 5 provides qualitative examples of our matching framework for a positive and negative sample. The first row provides the top to bottom matching whereas the second row shows the opposite, from the bottom to the top. Notice that in the positive sample case, both directions are much more consistent than in the negative sample case.

6.2. Experimental comparison to GMN

Among the graph metric learning approaches in the literature, the graph matching networks (GMN) work [36] is the most prominent one. In this section, we propose an extra experiment to compare with their work.

6.2.1. Dataset Description

The IAM Graph Database Repository [64] provides several graph datasets covering a wide spectrum of different applications. In particular, we focused on

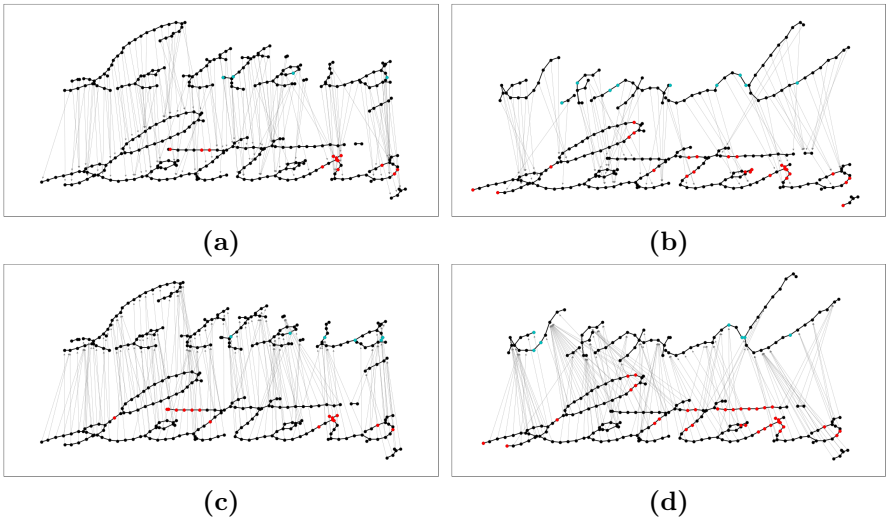


Figure 5: Visualization of the learned node correspondence. First row, shows the node matching from top to bottom; the second row of the figure shows de opposite. (a)-(d) "Letters"–"Letters"; (b)-(d) "send"–"Letters"

the COIL-DEL dataset. The COIL-100 [65] consists of 100 object images at different poses. In order to construct the COIL-DEL dataset, these images were converted into mesh graphs by means of the Harris corner detection algorithm followed by a Delaunay triangulation. COIL-DEL is divided in $2,400$, 500 and $1,000$ graphs for training, validation and test respectively. In average these graphs have 21.5 nodes and 54.2 edges. Thus, they are rather small graphs.

6.2.2. Experimental protocol

In these experiments, we followed the same experimental protocol introduced by Li *et al.* [36], so we evaluated our method on two different metrics:

- **pair AUC**: The area under the ROC curve for classifying pairs of graphs as similar or not on a fixed set of 1,000 pairs.
- **triplet accuracy**: The accuracy of correctly assigning a higher similarity to the positive pair than a negative pair on a fixed set of 1,000 triplets.

Note that the fixed pairs and triplets are not the same from the original paper. We performed a random selection of these pairs and triplets while trying

to balance the number of examples per class.

6.2.3. Results and Discussion

Table 5 presents a comparison to the state-of-the-art techniques on graph metric learning. In particular, we compare with the different architectures proposed in [36].

It is not surprising that their GMN technique outperforms our proposed model as they do incorporate cross-graph connections following an attention paradigm. Therefore, the correspondence is learned end-to-end in a much robust way. However, this is only feasible in rather small datasets as it incorporates a huge computational overhead. Notice also, that their GNN and Siamese-GNN models just obtain a slightly better performance than our proposed approach. However, when dealing with such small graphs it is hard to compare against embedding based approaches as they are able to encode the graph characteristic features without a huge loss of information.

In this extra experiment, we have also evaluated the effect on the choice of GNN models, the number of layers and the margin parameter μ. From this table, we conclude that the GRU-based models are drivers of a better performance on these experiments. Moreover, we find important to set our margin parameter to 1. The number of layers has not proven to bring a boost on performance on this particular dataset. Overall, our best model is able to obtain comparable results against GMN in this small dataset.

7. Conclusions and Future Work

In this paper, we have proposed a triplet GNN architecture for learning graph distances. Our architecture is able to learn node embeddings based on structural information of nodes local contexts. These learned features lead to an enriched graph representation which is later used in the distance computation. Moreover, we extended the graph edit distance approximation *viz.* Hausdorff edit distance, to the new learning framework in order to learn its operation costs within an end-to-end fashion. We have validated our proposed architecture

Table 5: Performance comparison on the COIL-DEL dataset against the methodologies introduced in [36]. We studied the effect of the proposed GAT and GRU models, as well as, the number of layers and margin parameter μ.

Model		# Layers (K)	μ	Pair AUC	Triplet Acc
GCN [36]		-	-	94.80	94.95
Siamese-GCN [36]		-	-	95.90	96.10
GNN [36]		-	-	98.58	98.70
Siamese-GNN [36]		-	-	98.76	98.55
GMN [36]		-	-	98.97	98.80
Our	GAT	3	1	97.82	96.74
			10	96.22	96.94
		5	1	97.85	96.94
			10	97.70	97.54
	GRU	3	1	98.08	97.50
			10	96.25	95.69
		5	1	97.56	97.60
			10	95.36	96.07

on a graph retrieval scenario, in particular, we faced a keyword spotting task for handwritten words. Finally, we demonstrated competitive results against state-of-the-art, learning-based methods for graph distance learning.

Several future research lines emerge taking our proposed framework as starting point. For instance, our framework does not exploit the edge structure at matching time as we considered it implicitly encoded as node features. However, leveraging the edges information at this stage might lead to better results. Another promising line of research relates to the use of different graph pooling layers for reducing the size of large graphs before computing the learned Hausdorff edit distance.

Acknowledgment

This work has been partially supported by the Spanish project RTI2018-095645-B-C21, the FPU fellowship FPU15 / 06264, the Ramon y Cajal Fellowship RYC-2014-16831, and the CERCA Program / Generalitat de Catalunya. We thank NVIDIA for the donation of a Titan Xp GPU.

References

References

[1] J. Yang, J. Lu, S. Lee, D. Batra, D. Parikh, Graph r-cnn for scene graph generation, in: Proceedings of the European Conference on Computer Vision, 2018.

[2] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, G. E. Dahl, Neural message passing for quantum chemistry, in: Proceedings of the International Conference on Machine Learning, 2017, pp. 1263–1272.

[3] G. Gkioxari, J. Malik, J. Johnson, Mesh r-cnn, in: Proceedings of the IEEE International Conference on Computer Vision, 2019, pp. 9785–9795.

[4] D. Conte, P. Foggia, C. Sansone, M. Vento, Thirty years of graph matching in pattern recognition, International Journal of Pattern Recognition and Artificial Intelligence 18 (3) (2004) 265–298.

[5] P. Foggia, G. Percannella, M. Vento, Graph matching and learning in pattern recognition in the last 10 years, International Journal of Pattern Recognition and Artificial Intelligence 28 (1) (2014) 1–40.

[6] K. M. Borgwardt, Graph kernels, Ph.D. thesis, Ludwig-Maximilians-Universität München (2007).

[7] F. Zhou, F. de la Torre, Factorized graph matching, IEEE Transactions on Pattern Analysis and Machine Intelligence 99 (2015) 1–1.

[8] J. Gibert, E. Valveny, H. Bunke, Graph embedding in vector spaces by node attribute statistics, Pattern Recognition 45 (9) (2012) 3072–3083.

[9] A. Dutta, P. Riba, J. Lladós, A. Fornés, Hierarchical stochastic graphlet embedding for graph-based pattern recognition, Neural Computing and Applications 32 (2020) 11579–11596.

[10] R. Kondor, H. Pan, The multiscale laplacian graph kernel, in: Advances in Neural Information Processing Systems, 2016, pp. 2982–2990.

[11] A. Sanfeliu, K.-S. Fu, A distance measure between attributed relational graphs for pattern recognition, IEEE transactions on systems, man, and cybernetics (3) (1983) 353–362.

[12] H. Bunke, G. Allermann, Inexact graph matching for structural pattern recognition, Pattern Recognition Letters 1 (4) (1983) 245–253.

[13] X. Gao, B. Xiao, D. Tao, X. Li, A survey of graph edit distance, Pattern Analysis and applications 13 (1) (2010) 113–129.

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

[15] A. Fischer, C. Y. Suen, V. Frinken, K. Riesen, H. Bunke, Approximation of graph edit distance based on Hausdorff matching, Pattern Recognition 48 (2) (2015) 331–343.

[16] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, in: Advances in Neural Information Processing Systems, 2012, pp. 1097–1105.

[17] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, P. Vandergheynst, Geometric deep learning: going beyond euclidean data, IEEE Signal Processing Magazine 34 (4) (2017) 18–42.

[18] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, R. Shah, Signature verification using a "siamese" time delay neural network, in: Advances in Neural Information Processing Systems, 1994, pp. 737–744.

[19] K. Q. Weinberger, L. K. Saul, Distance metric learning for large margin nearest neighbor classification, Journal of Machine Learning Research 10 (2009) 207–244.

[20] I. Elezi, S. Vascon, A. Torcinovich, M. Pelillo, L. Leal-Taixe, The group loss for deep metric learning, arXiv preprint arXiv:1912.00385.

[21] T. Chen, S. Kornblith, M. Norouzi, G. Hinton, A simple framework for contrastive learning of visual representations, arXiv preprint arXiv:2002.05709.

[22] P. Riba, A. Fischer, J. Lladós, A. Fornés, Learning graph distances with message passing neural networks, in: Proceedings of the International Conference on Pattern Recognition, 2018, pp. 2239–2244.

[23] M. Gori, G. Monfardini, F. Scarselli, A new model for learning in graph domains, in: IEEE International Joint Conference on Neural Networks, Vol. 2, 2005, pp. 729–734.

[24] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, G. Monfardini, The graph neural network model, IEEE Transactions on Neural Networks 20 (1) (2009) 61–80.

[25] J. Bruna, W. Zaremba, A. Szlam, Y. LeCun, Spectral networks and locally connected networks on graphs, arXiv preprint arXiv:1312.6203.

[26] M. Henaff, J. Bruna, Y. LeCun, Deep convolutional networks on graph-structured data, arXiv preprint arXiv:1506.05163.

[27] M. Defferrard, X. Bresson, P. Vandergheynst, Convolutional neural networks on graphs with fast localized spectral filtering, in: Advances in Neural Information Processing Systems, 2016, pp. 3844–3852.

[28] T. N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, Proceedings of the International Conference on Learning Representations.

[29] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Bengio, Graph attention networks, Proceedings of the International Conference on Learning Representations.

[30] Y. Li, D. Tarlow, M. Brockschmidt, R. Zemel, Gated graph sequence neural networks, in: Proceedings of the International Conference on Learning Representations, 2016, pp. 1–20.

[31] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, M. Sun, Graph neural networks: A review of methods and applications, arXiv preprint arXiv:1812.08434.

[32] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, et al., Relational inductive biases, deep learning, and graph networks, arXiv preprint arXiv:1806.01261.

[33] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, S. Y. Philip, A comprehensive survey on graph neural networks, IEEE Transactions on Neural Networks and Learning Systems.

[34] V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, X. Bresson, Benchmarking graph neural networks, arXiv preprint arXiv:2003.00982.

[35] P. Baldi, Y. Chauvin, Neural networks for fingerprint recognition, Neural Computation 5 (3) (1993) 402–418.

[36] Y. Li, C. Gu, T. Dullien, O. Vinyals, P. Kohli, Graph matching networks for learning the similarity of graph structured objects, in: Proceedings of the International Conference on Machine Learning, 2019, pp. 3835–3845.

[37] U. Chaudhuri, B. Banerjee, A. Bhattacharya, Siamese graph convolutional network for content based remote sensing image retrieval, Computer Vision and Image Understanding 184 (2019) 22–30.

[38] J. Zhang, Graph neural distance metric learning with graph-bert, arXiv preprint arXiv:2002.03427.

[39] Y. Bai, H. Ding, S. Bian, T. Chen, Y. Sun, W. Wang, Simgnn: A neural network approach to fast graph similarity computation, in: Proceedings

of the Twelfth ACM International Conference on Web Search and Data Mining, 2019, pp. 384–392.

[40] Y. Bai, H. Ding, K. Gu, Y. Sun, W. Wang, Learning-based efficient graph similarity computation via multi-scale convolutional set matching, in: Proceedings of the AAAI Conference on Artificial Intelligence, 2020.

[41] A. Sanfeliu, K. Fu, A distance measure between attributed relational graphs for pattern recognition, IEEE Transactions on Systems, Man, and Cybernetics SMC-13 (3) (1983) 353–362.

[42] V. Levenshtein, Binary codes capable of correcting deletions, insertions, and reversals, Soviet Physics Doklady 10 (8) (1966) 707–710.

[43] R. A. Wagner, M. J. Fischer, The string-to-string correction problem, Journal of the ACM 21 (1) (1974) 168–173.

[44] Z. Zeng, A. K. Tung, J. Wang, J. Feng, L. Zhou, Comparing stars: On approximating graph edit distance, Proceedings of the VLDB Endowment 2 (1) (2009) 25–36.

[45] D. Justice, A. Hero, A binary linear programming formulation of the graph edit distance, IEEE Transactions on Pattern Analysis and Machine Intelligence 28 (8) (2006) 1200–1214.

[46] M. Neuhaus, K. Riesen, H. Bunke, Fast suboptimal algorithms for the computation of graph edit distance, in: Structural, Syntactic, and Statistical Pattern Recognition, 2006, pp. 163–172.

[47] J. Munkres, Algorithms for the assignment and transportation problems, Journal of the society for industrial and applied mathematics 5 (1) (1957) 32–38.

[48] F. Serratosa, Fast computation of bipartite graph matching, Pattern Recognition Letters 45 (2014) 244 – 250.

[49] A. Fischer, S. Uchida, V. Frinken, K. Riesen, H. Bunke, Improving hausdorff edit distance using structural node context, in: Proceedings of the Workshop on Graph-Based Representation in Pattern Recognition, 2015, pp. 148–157.

[50] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, in: Proceedings of the International Conference on Machine Learning, 2015, pp. 448–456.

[51] K. Cho, B. Van Merriënboer, D. Bahdanau, Y. Bengio, On the properties of neural machine translation: Encoder-decoder approaches, in: Workshop on Syntax, Semantics and Structure in Statistical Translation, 2014.

[52] V. Garcia, J. Bruna, Few-shot learning with graph neural networks, in: Proceedings of the International Conference on Learning Representations, 2018.

[53] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980.

[54] D. P. Vassileios Balntas, Edgar Riba, K. Mikolajczyk, Learning local feature descriptors with triplets and shallow convolutional neural networks, in: Proceedings of the British Machine Vision Conference, 2016, pp. 119.1–119.11.

[55] M. Stauffer, A. Fischer, K. Riesen, A novel graph database for handwritten word images, in: Joint International Workshops on Statistical techniques in Pattern Recognition and Structural and Syntactic Pattern Recognition, 2016, pp. 553–563.

[56] M. Stauffer, A. Fischer, K. Riesen, Keyword spotting in historical handwritten documents based on graph matching, Pattern Recognition 81 (2018) 240–253.

[57] A. Fischer, A. Keller, V. Frinken, H. Bunke, Lexicon-free handwritten word spotting using character hmms, Pattern Recognition Letters 33 (7) (2012) 934–942.

[58] I. Pratikakis, K. Zagoris, B. Gatos, J. Puigcerver, A. H. Toselli, E. Vidal, Icfhr2016 handwritten keyword spotting competition (h-kws 2016), in: Proceedings of the International Conference on Frontiers in Handwriting Recognition, 2016, pp. 613–618.

[59] M. R. Ameri, M. Stauffer, K. Riesen, T. D. Bui, A. Fischer, Graph-based keyword spotting in historical manuscripts using hausdorff edit distance, Pattern Recognition Letters 121 (2019) 61–67.

[60] M. Rusiñol, J. Lladós, A performance evaluation protocol for symbol spotting systems in terms of recognition and location indices, International Journal on Document Analysis and Recognition 12 (2) (2009) 83–96.

[61] S. Sudholt, G. A. Fink, Phocnet: A deep convolutional neural network for word spotting in handwritten documents, in: Proceedings of the International Conference on Frontiers in Handwriting Recognition, 2016, pp. 277–282.

[62] J. Almazán, A. Gordo, A. Fornés, E. Valveny, Word spotting and recognition with embedded attributes, IEEE Transactions on Pattern Analysis and Machine Intelligence 36 (12) (2014) 2552–2566.

[63] T. Wilkinson, A. Brun, Semantic and verbatim word spotting using deep neural networks, in: Proceedings of the International Conference on Frontiers in Handwriting Recognition, 2016, pp. 307–312.

[64] K. Riesen, H. Bunke, Iam graph database repository for graph based pattern recognition and machine learning, in: Joint International Workshops on Statistical techniques in Pattern Recognition and Structural and Syntactic Pattern Recognition, 2008, pp. 287–297.

[65] S. Nayar, S. Nene, H. Murase, Columbia object image library (coil 100), Dept. of Comp. Science, Columbia University, Tech. Rep. CUCS-006-96.