Pattern Recognition 144 (2023) 109867

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/pr

Adaptive filters in Graph Convolutional Neural Networks

Andrea Apicella [a,b,c,*], Francesco Isgrò [a,b,c], Andrea Pollastro [a,b,c,d], Roberto Prevete [a,b,c]

[a] Laboratory of Artificial Intelligence, Privacy & Applications (AIPA Lab), University of Naples Federico II, Italy
[b] Laboratory of Augmented Reality for Health Monitoring (ARHeMLab), University of Naples Federico II, Italy
[c] Department of Electrical Engineering and Information Technology, University of Naples Federico II, Italy
[d] Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA

ARTICLE INFO

Keywords:
Graph Convolutional Neural Networks
Deep learning
Dynamic neural networks
Programmable ANNs
Graph structure learning

ABSTRACT

Over the last few years, the availability of an increasing data generated from non-Euclidean domains, which are usually represented as graphs with complex relationships, and Graph Neural Networks (GNN) have gained a high interest because of their potential in processing graph-structured data. In particular, there is a strong interest in performing convolution on graphs using an extension of the GNN architecture, generally referred to as Graph Convolutional Neural Networks (ConvGNN). Convolution on graphs has been achieved mainly in two forms: spectral and spatial convolutions. Due to the higher flexibility in exploring and exploiting the graph structure of data, there is recently an increasing interest in investigating the possibilities that the spatial approach can offer. The idea of finding a way to adapt the network behaviour to the inputs they process to maximize the total performances has aroused much interest in the neural networks literature over the years. This paper presents a novel method to adapt the behaviour of a ConvGNN to the input performing spatial convolution on graphs using input-specific filters, which are dynamically generated from nodes feature vectors. The experimental assessment confirms the capabilities of the proposed approach, achieving satisfying results using a low number of filters.

1. Introduction

In the last few decades, Convolutional Neural Networks (CNNs) have gained much interest due to their potential and versatility in addressing a large scale of machine learning and pattern recognition problems [1], while achieving great success. The potential of CNNs lies in extracting and processing local information performing convolution on input data using sets of trainable filters with a fixed size. However, the design of the convolution operation in the CNNs allows to process only regular data while, in the real world, there is a considerable amount of data that naturally lie on non-Euclidean domains, needing different techniques to be processed. These data are often represented by *graph-based* structures. Graph structures imply several difficulties in using standard data processing techniques, such as the impossibility of using classic CNNs due to the variable number of neighbours for each node (differently from regular data where the filter properties fix the number of neighbours for each node). This aspect has led to new processing techniques such as Graph Neural Networks (GNNs), which gained high interest during the last years. First attempts [2] of neural networks based on input graphs, generally referred to as Recurrent Graph Neural Networks (RecGNNs), were based on *message passing* architectures, where an iterative process allows to learn, for each node, a representation of the relative neighbourhood information. Therefore, the learned node representations are used for classification or regression tasks.

Due to the great success of CNNs, GNNs inherits convolution operation producing the Graph Convolutional Neural Networks (ConvGNNs), which have found their expression in two different approaches. The former are *spectral* methods (see, for example, [3]), that perform convolution based on graphs signal processing techniques. The latter are *spatial* methods (see, for example, [4]), that instead perform convolution using spatial information of data, similarly to what classical CNNs do. ConvGNNs share the same idea of message passing with RecGNNs but in a non-iterative manner. However, ConvGNNs are usually based on learned filters having constant values for each input fed to the network as well as classical convolutional networks. In other words, the filter values are independent of the input values. However, we note that adapting the Artificial Neural Network (ANN) inner behaviour in function of the input is an open research area in the scientific community. In a nutshell, in standard approaches, the ANN input–output relationship, i.e., the ANN behaviour, after the training phase, is completely defined by a set of fixed network parameters (weights and biases). By contrast, the core idea of this research area is that the ANN behaviour also depends on the input itself or additional inputs. A way to achieve this goal is setting the network parameters by another neural

* Corresponding author at: Department of Electrical Engineering and Information Technology, University of Naples Federico II, Italy.
E-mail address: andrea.apicella@unina.it (A. Apicella).

https://doi.org/10.1016/j.patcog.2023.109867
Received 21 March 2022; Received in revised form 6 July 2023; Accepted 31 July 2023
Available online 5 August 2023

network which receives the same input of the former neural network or external/additional inputs (see, for example, [5]). Thus, ANN is able to dynamically change its behaviour according with the received inputs. In this paper, we refer to this type of approaches as *Dynamic Behaviour Neural Networks* (DBNN).

In the last years, several works were proposed following DBNN approach [6]. However, in the GNN field, to the best of our knowledge, the DBNN approach has not received too much attention. In [7] the authors proposed the Edge-Conditioned Convolution (ECC) network, which performs spatial convolutions over graph neighbourhoods exploiting edge labels and generating input-specific filters from them. In [8], the authors proposed a method to make the GNN message passing architecture adaptive to input nodes using an external hypernetwork. Zhang et al. in [9] proposed the Graph HyperNetwork (GHN), a model having weights generated by an hypernetwork exploiting a computation graph representation. In [10], an hypernetwork architecture is proposed to generate relation-specific convolutional filters for convolution on graphs.

In this paper, we exploit the possibility of dynamically changing the convolutional filter behaviour as a function of the input and propose a novel method to perform spatial convolution on graph-structured data. We will name our approach Dynamic Graph Convolutional Filters (DGCF). Following [7,11], convolutional filters will be generated using an external module, the *filter-generating network*, that, during the training stage, learns to produce input-specific filters in order to perform an *ad-hoc* filtering operation for each input sample. The proposed approach allows a network to change its behaviour according to the input. Indeed, the convolutional filters are dynamically generated using an external module based on the input graphs. This brings two main benefits. The former is that we obtain a low number of convolutional filters since the model can build the most suitable filter according to the network input instead of relying on a battery of fixed filters learned during the learning phase. In this work, we show this empirically by reporting a visual analysis of the training stages, inspecting differences in filter generation according to the input. We show that specific filters are obtained according to the input class, enhancing the main architecture by intrinsic patterns hidden in the input itself. The latter benefit of this type of approach is that the training phase involves a smaller network to generate the weights of another one, resulting in a more efficient training phase. In fact, in our experiments, we noticed that the final training convergence is reached in fewer epochs than in classical approaches.
Differently from [7], in our approach filters are generated exploiting nodes feature vectors instead of edge labels. Our approach is validated in three series of experiments, making a comparison with the standard convolution over 10 repetitions, with randomly initialized weights for each repetition. Hypothesis tests are reported for each series of experiment to verify the significance of the results. The advantages of the proposed approach can be summarized as follows: (i) it inherits the standard convolution operation from classical CNNs. The convolution is performed on each node with its nearest neighbours applying sets of fixed-sized filters; (ii) the network inner behaviour changes according to the input. Filters used for the convolution are dynamically generated using an external module based on the input graphs; (iii) dynamic behaviour of the convolutional filters can lead to design simpler architectures than non-dynamic approaches with respect the number of convolutional filters, while leaving unaltered performances. Promising results can be achieved using a fewer number of convolutional filters than a non-dynamic approach; (iv) training convergence can be reached in a fewer number of epochs. Using a dynamic approach, we empirically show that the learning stage needs a fewer number of epochs than using the non-dynamic approach.

This paper is organized as follows. Section 2 briefly reviews the related literature; Section 3 describes the proposed method; the experimental assessment is described in Section 4 while in Section 5 the obtained results are presented and discussed. In Section 6 a visual analysis of the training stage of our proposal is shown. In Section 7 results and outcomes of a study are discussed. The concluding Section 8 is left to final remarks.

2. Related works

In this section, we first report the related works in the context of DBNN approach and, then, we give a general description of the Graph Neural Networks, focusing on ConvGNNs.

2.1. DBNN approaches

The idea of controlling the behaviours of an ANN through the input itself or an additional/external input has a long history in the literature For example, in [12], the authors describe a way to represent general conditional probability densities by considering a parametric model for the distribution expressed as a neural network whose parameters are determined by the outputs of another neural network having the same input. In [6] the filters of CNNs and LSTMs networks are generated by an auxiliary network. In [11] the authors defined the dynamic changes in ANNs' behaviours in the context of traditional CNNs using a proposed *dynamic filter module* to execute the convolution operation. Dynamic filter module consists of two parts: a *filter-generating network*, that generates filters' parameters from a given input, and a *dynamic filtering layer*, that applies those generated filters to another input. In particular, the dynamic filtering layer can be instantiated as a *dynamic convolutional layer*, wherever the filtering operation is translation invariant. In [11], considering two input images I_A and I_B, not necessary different, the filter-generating network takes as input I_A and outputs filters $\mathcal{F}_\theta$ to apply on I_B. Filters $\mathcal{F}_\theta$ are parameterized by parameters θ. In this way, an output $G = \mathcal{F}_\theta(I_B)$ is generated. However, this method is developed in the context of classic CNNs; by contrast, in [7], the authors attempt to perform a dynamic spatial convolution on graphs. The authors proposed the Edge-Conditioned Convolution (ECC), which uses a filter-generating network to output edge-specific filters for each input sample dynamically.

The DGCF approach proposed in this paper is inspired by the work in [7,11]. We perform a convolution on input graphs using filters that are dynamically generated by a filter-generating network, thus obtaining a dynamical change in the behaviour of the ConvGNN. We point out that, differently from the ECC proposed in [7] where convolutional filters are *edge-based*, our strategy (see Section 3) considers *node-based* filters, tweaking in this way the filtering operation on nodes by nodes themselves. Thus, summarizing, the proposed approach is different from ECC with respect to the input fed to the filter-generating network, and it differs from the other DBNN approaches since it is applied on graphs.

2.2. Graph Neural Networks

GNNs are showing positive effects in several applications, as for example in time-series forecasting tasks [13], document analysis [14], and image analysis [15–18]. A first proposal of a neural network model for graph-structured data was made in [2]. This model builds on the idea that graph nodes represent "concepts" related to each other via edges. Each node n is represented by a feature vector x_n and each edge (i,j) is described by a feature vector $\mathbf{x}^e_{(i,j)}$. This model leverages information exchange among nodes and their neighbours to update their features iteratively (*message passing* mechanism). In the literature, iterative graph processing techniques based on neural networks with a message passing architecture are generally referred as RecGNNs. To face the computational costs of these methods, several kinds of neural network models were proposed in the literature, often with the aim of generalizing classical and established neural networks data processing to graph data. In [19] a comprehensive survey on the topic is proposed.

A particular focus was given in the literature to perform the convolution operation on graph-structured data. Graph Convolutional Neural Networks (ConvGNNs) share the idea of message passing adopted by RecGNNs but implement it in a non-iterative manner: information is exchanged between neighbours using different convolutional layers, each with different filters [19]. However, the non-Euclidean characteristics of graphs (e.g., their irregular structure) makes the convolution and filtering operations not easy to define as for those on images. For this reason, in the past decades, researchers have been working on how to conduct convolution operations on graphs using several approaches, that can be categorized in (i) *spectral approaches*, that rely on the graph spectral theory, involving graph signal processing, such as graph filtering and graph wavelets (see, for example, [20]), and (ii) *spatial approaches*, that leverage on structural information to perform convolution, such as aggregations of graph signals within the node neighbourhood (see, for example, [4,21]). Although spectral architectures have been explored successfully in several works (see, for example, [20]), one of the main problems of ConvGNNs in the spectral domain is that the graph structure has to be set for all the inputs due to the use of the graph Laplacian in the training stage. However, spectral analysis is computationally expensive, limiting the concrete usage of this methods on huge graphs [22]. Indeed, according to [19], the first proposals on spectral GNN [23,24] require eigenvalue decomposition of the graph matrix. Therefore they have a computational complexity of $O(n^3)$, where n is the number of graph nodes. Instead, first spatial approaches [21,25] result in a computational complexity of $O(m)$ and $O(n^2)$ respectively, where m is the total number of edges in the graph. However, a spectral approach based on the Chebyshev polynomials to reduce the computational complexity is proposed in [26] and further simplified in [27]. In this case, the computational complexity is $O(m)$, but the algorithms make several approximations and simplifications. Although strategies to use ConvGNNs with different inputs graph structures were proposed [20], this problem is generally not present in the spatial domain. For these reasons, several spatial domain methods have been proposed in the literature. For example, in [28] the authors present PATCHY-SAN, a ConvGNN model inspired by the classical image-based CNN. In [21] and [4] two different methods to generalize the convolutional operator using random walks for neighbourhood locating were reported. In [29] spectral-based GNNs are generalized to work on data structured as hypergraphs [30] instead of classical graphs. In [31] the complexity of a GNN was reduced, collapsing the network layers into a single linear transformation. In [32] the authors proposed a method to learn or refine the graph structure together with the network parameters.

The aim of the work presented in this paper is to perform an adaptive spatial-convolution, using fixed-sized filters, on graph structured data. According to the dynamic convolutional layer proposed in [11], in our approach, for each sample, a translation invariant set of filters is generated by a filter-generating network and shared among all the neighbourhoods.

3. Method description

In this work, we propose a ConvGNN-based architecture whose convolutional filters change in function of the input features. Differently from similar works such as [7], where filters depend on the graph edges, we propose a DBNN approach based on the graph nodes' features. In this section, after a brief introduction of graphs' notation, a detailed description of our proposal is given.

3.1. Notations

Let $G = (V, E)$ be an undirected or directed graph where V is a finite set of N nodes, and E is a finite set of edges. We define in boldface $\mathbf{x}^i \in \mathbb{R}^{1\times J}$ the input feature vector related to the node $i \in V$, where J is the number of input channels, and $\mathbf{y}^\mathbf{i} \in \mathbb{R}^{1\times M}$ its output feature vector, where M is the number of output channels. Let $X \in \mathbb{R}^{N\times J}$ denote the matrix representation of an input graph as an embedding of the feature vectors of its nodes. In order to obtain neighbourhoods with a sufficient number of nodes to which apply a filter of dimension K, we select the *k-nearest neighbours* of each node using the classical shortest path distance. Neighbourhoods are uniquely defined for each node.

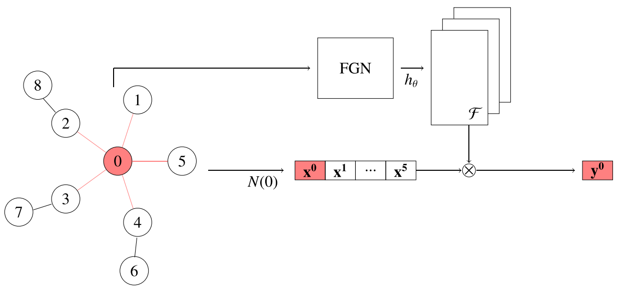


Fig. 1. A graphical description of the DGCF in processing a node – marked in red and labelled as 0 – using filters of dimension $K = 5$. A unique set of filters dependent on all the nodes' feature vectors of the input graph is dynamically generated by the Filter Generating Network (FGN) and shared among all of its neighbourhoods. Finally, input feature vectors of the neighbourhoods are weighted summed to compute the nodes' outputs. In this example, input-specific filters are applied on the neighbourhood of the node 0, referred as $N(0)$. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

3.2. Dynamic Graph Convolutional Filters

This work aims to perform convolution on graphs using dynamically generated filters conditioned on a given input. As we have described above, otherwise from ECC in [7], where convolution is performed using dynamical *edge-based* filters, our intent consists in using *node-based* filters dynamically generated from nodes' feature vectors.

Considering the matrix representation X of an input graph, using a neural network $h_\theta(\cdot)$, that we will refer as filter-generating network, with parameters θ, we can generate a set of node-specific filters $\mathcal{F} = h_\theta(X)$ used to compute a dynamic convolution on input graphs. $\mathcal{F}$ can be represented as a matrix $\mathcal{F} \in \mathbb{R}^{J\times K\times M}$, where J is the number of input channels, K is the filter size and M is the number of output channels. Supposing to compute the mth output channel of the node n, what we propose can be formalized as follows:

$$\mathbf{y}_m^n = f\left(\sum_{j=1}^{J}\sum_{k=1}^{K}\mathcal{F}_{jkm}\mathbf{x}_j^{s(n,k)}\right) \tag{1}$$

where $s(n,k)$ returns the index of the kth neighbour of n, $\mathcal{F}J_{jkm}$ are the filters generated by the filter-generating network $h_\theta(\cdot)$. In other words, the $\mathcal{F}_{jkm}$ is the value in position (j,k) of the mth filter generated by the filter-generating network $h_\theta(\cdot)$. Thus, during the training stage, the parameters θ have to be learned, together with the other network's parameters. According to what is described in [11], our approach follows the *dynamic convolutional layer*: the filter-generating network, defined as $h_\theta : \mathbb{R}^{N\times J} \to \mathbb{R}^{J\times K\times M}$, where N is the number vertices, J is the number of input channels, K is the filter size and M is the number of the output channels, takes as input the input graph and generates a unique set of filters shared among all the neighbourhoods.

A graphical representation of the proposed DGCF model is shown in Fig. 1: supposing to have a node, labelled as 0, as a central node during the convolution operation on a given input graph, an input-specific set of filters is firstly generated by the filter-generating network using the nodes' feature vectors of the input graph, then it is applied to the neighbourhood of the node 0, referred to as $N(0)$, computing a new representation of it. This procedure is then iterated over all the nodes of the input graph. From the experimental results, as reported in Section 5, emerged that the use of few dynamic convolutional filters leads to results comparable with traditional convolutional architectures composed of an higher number of filters.

3.3. Computational complexity

Considering (1), the time required by $\mathbf{y}_m^n$ depends on the time T_{FGN} required by a forward pass on the Filter-Generating Network $h_\theta(\cdot)$ to

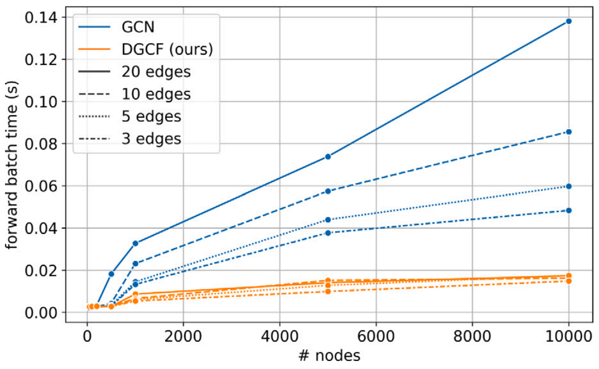


Fig. 2. The time required by the proposed DGCF as the number of graph nodes increases, considering several numbers of edges for each node. As a comparison, the time needed by GCN [27] on the same data is also reported. The simulation was made using 28 artificially generated datasets, composed of 1000 graphs with nodes $n \in \{10000, 5000, 1000, 500, 200, 100, 50\}$, edges $e \in \{20, 10, 5, 3\}$, and 32 channels. The Architectures are composed of 2 convolutional layers with 16 and 8 output channels respectively. Times were computed averaging on batches of size 32.

compute $\mathcal{F}_{jkm}$. This time depends on the $h_\theta(\cdot)$ architecture. Considering FGN composed of fully-connected layers, the time is linear on the number of its weights W, i.e. $O(W)$. Therefore, the computation of $\mathbf{y}^n$ for a single node n over all the $1 \leq m \leq M$ channels, has a time complexity of $O(W + M \cdot J \cdot K)$. The time complexity for computing the output of the full layer. An empirical analysis on the time required by the proposed DGCF as the number of graph nodes increases is shown in Fig. 2.

4. Experimental assessment

The DGCF approach was experimentally evaluated in three series of experiments. We point out that our interest in these experiments is investigating the advantages of using our method to non-dynamic approaches in terms of results, learning time, and model simplicity. At first, we conducted preliminary experiments on the well-known benchmark dataset MNIST, representing images in terms of graphs [4][33]. Then, we conducted a series of experiments on the 20NEWS,[1] a commonly used dataset in the GNN literature [32,34,35]. Finally, we conducted a series of experiments on an widely used electroencephalographic (EEG) signals dataset, SEED [36], in order to test our method on harder tasks, such as emotion recognition from EEG signals. On this last task, a model analysis was made in order to investigate the functioning of DGCF approach. In particular, a comparison between the learning processes (in terms of weights updates for each epoch) between the static filters and the filters generated by the filter-generating network was carried out. Table 1 reports a summary of the datasets used in our experiments. Note that, in each series of experiments, we selected as comparison the non-dynamic ConvGNN [4] reported in the literature, trained on the same dataset we used.

4.1. MNIST

We used the widely known MNIST handwritten digits dataset for running the first series of experiments. The MNIST dataset consists in 70,000 grayscale images of handwritten digits. This dataset was reported already split in training (60,000 samples) and test (10,000 samples) sets. So, recognizing each digit can be viewed as a 10-classes classification problem. We adopted the same experimental setup on the MNIST dataset used in [4] that can be resumed as follows: after the

[1] http://qwone.com/~jason/20Newsgroups/

Table 1
Datasets selected for the experiments.

Dataset	*#graphs*	*#nodes*	*#classes*
MNIST	70000	717	10
20NEWS	17236	10000	20
SEED	50910	62	3

exclusion of constant pixels, the correlation matrix C between pixels is computed to estimate their relationships. Therefore, two pixels (nodes) i, j are considered connected if $|C_{ij}| > T$ where T is a fixed threshold. In this work, the experiments were made with a threshold value of $T = 0.5$.

4.2. 20NEWS dataset

The 20NEWS dataset consists of about 20,000 text documents, labelled using 20 classes. The original split in training set (10,167 samples after the preprocessing) and test set (7069 samples after the preprocessing) was adopted in this work. This task can be defined as a 20-classes classification problem. Following the preprocessing described in [32], we considered only the 10,000 most frequently used words, considering the bag-of-words model to represent each document. We used *word2vec* [37] to represent each word as a vector. Finally, the cosine similarity metric between words vectors was adopted to compute the connections between words.

4.3. SEED

The SEED dataset [36] consists of EEG signals recorded from 15 subjects while they were watching video clips of about 4 min. Each video clip was carefully chosen to induce three types of emotions, i.e. negative, neutral and positive. For each subject, three sessions of 15 trials/video clips were collected. EEG signals were recorded in 62 channels using the ESI Neuroscan System.[2] As in [38], we consider the pre-computed differential entropy (DE) features smoothed by linear dynamic systems (LDS). DE features are pre-computed, for each second, in each channel, over the following five bands: delta (1–3 Hz); theta (4–7 Hz); alpha (8–13 Hz); beta (14–30 Hz); gamma (31–50 Hz). Samples were modelled as graphs considering each EEG channel as a node, and the DE features related to the 5 bands as feature vector of each node. The adjacency matrix $A \in \mathbb{R}^{n \times n}$ was modelled considering the EEG channels disposition on the scalp, where n represents the number of channels in EEG signals. In particular, each entry A_{ij} represents the physical distance between the sensor i and the sensor j.

4.4. Experimental setup

The proposal was validated by analysing its impact on architectures composed of two layers followed by a linear classifier, as already proposed in literature [4,32,38]. For each architecture, we evaluated the models as the number of convolutional filters changes, as it is shown in the relative configuration (Table 2). For the experiments on the MNIST and 20NEWS datasets, model performance estimation was performed using an *holdout method* since both of the datasets report a predefined train/test split. Moreover, model performances were evaluated based on the performances averaged over 10 repetitions, where, for each repetition, models' parameters were reinitialized following the same initialization criterion. For the experiments on the SEED dataset, model performance estimation was estimated focusing on the *Subject-Independent Classification*: still following the experimental protocol of [38], we adopted the *leave-one-subject-out cross-validation*, in which 14 subjects are considered as training set, and the remaining

[2] https://compumedicsneuroscan.com/

Table 2
Configurations used for the experiments on MNIST, 20NEWS, and SEED datasets. We denote with K the adopted filter sizes, with Ct a ConvGNN layer with t filters, with DCt a DGCF with t filters, and with FCn a fully connected layer with n hidden units.

	Conv. Layer architecture	FGN Architecture	K	Optimizer	Learning Rate
	MNIST				
DGCF (ours)	$DCt - C20, t \in \{2,4,8,16\}$	$FC200$	25	Adam	10^{-3}
ConvGNN [4]	$Ct - C20, t \in \{2,4,8,16\}$	–	25	Adam	Adam
	20NEWS				
DGCF (ours)	$DCt - FC100, t \in \{2,4,8,16\}$	$FC100 - FC100$	25	Adam	10^{-3}
ConvGNN [4]	$Ct - FC100, t \in \{2,4,8,16\}$	–	25	Adam	10^{-3}
	SEED				
DGCF (ours)	$DCt - AvgPool, t \in \{2,4,8,16,32\}$	$FC100 - FC100 - FC100$	9	Adam	10^{-3}
ConvGNN [4]	$Ct - AvgPool, t \in \{2,4,8,16,32\}$	–	9	Adam	10^{-3}

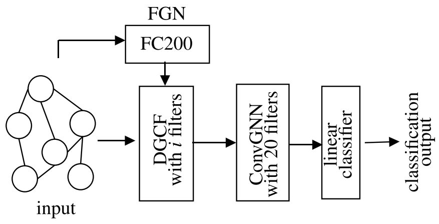


Fig. 3. Architecture adopted for experiments with MNIST dataset. See text and Table 2 for further details.

subject as test set. This was repeated for each possible configuration. The performance is evaluated averaging the test accuracies using one session of data. Moreover, for each experiment, during the training stage, 30% of the training set was extracted following a stratified sampling procedure. Each experiment was run considering *early stopping* as convergence criterion. Significance differences about the comparisons between the models were tested using hypothesis testing. For each result – expressed by the average and the standard deviation – a normality test was firstly made using the *Shapiro–Wilk* test. Then, according to the results of the normality tests, hypothesis tests were made using the Student's *t-test*, in the case of normally distributed data, or Mann–Whitney *U-test*, otherwise. For each test, a significance level of $\alpha = 0.05$ was considered. Hypothesis tests were formulated as follows:

$$H_0 : \mu_{ConvGNN} \geq \mu_{DGCF}$$

$$H_1 : \mu_{ConvGNN} < \mu_{DGCF}$$

Full-connected neural networks as filter-generating network (FGN) architectures were adopted, whose number of nodes and layers was tuned using a *Bayesian optimization* [39] in each experiment.

We ran the experimental assessment on a workstation with an NVIDIA 3080 GPU. The software was implemented in Python 3.7 adopting PyTorch 1.13 library. The code is freely available on GitHub.[3]

5. Results

In each experiment, the adopted models were two layers architectures [4,32,38] $L_1 - L_2$, $L_i \in \{Ct, DCt, FCn, AvgPool\}$ equipped with ReLU activation functions followed by a final linear classifier, where Ct, DCt referred to a t filters static ConvGNN layer and a t filters DGCF respectively, FCn referred to a fully connected layer having n hidden units, and $AvgPool$ referred to a global pooling layer. As FGNs, we adopted one, two, and three layers architectures equipped with ReLU for MNIST, 20 NEWS, and SEED experiments respectively. All the details about the architectures are reported in Table 2.

[3] https://github.com/andrea-pollastro/DGCF

Table 3
Hypothesis tests results for the MNIST dataset.

Hypothesis tests					
Configuration	$C2 - C20$	$C4 - C20$	$C8 - C20$	$C16 - C20$	$C20 - C20$
p-value	0.0152	0.0146	0.0066	0.0322	0.0420

Table 4
Hypothesis tests results for the 20NEWS dataset.

Hypothesis tests				
Configuration	$C2 - C20$	$C4 - C20$	$C8 - C20$	$C16 - C20$
p-value	0.0015	0.0010	0.0010	< 0.0001

5.1. MNIST

Using the MNIST dataset, the configuration $C20 - C20$ proposed by [4] is used. In the experimental setup of this work, a comparison was made substituting the first convolutional layer with a dynamic one. In our analysis, we varied the number of convolutional filters of the first layer in order to evaluate the effectiveness in using a dynamical approach, in both the static and dynamic models. The adopted architecture and parameters configurations are reported in Fig. 5 and in Table 2 respectively.

As it is shown in Fig. 4(a), the introduction of the dynamical layer increases the average performance of the architecture. Moreover, its interesting to notice that our dynamic approach leads to good performances also with simpler architectures: results comparable with the ones reported in [4] are achieved using a fewer number of convolutional filters. It is also important to point out that using our approach, convergence during the training stage was reached in a fewer number of epochs, as it is shown in Fig. 4(b). In Table 3 the results related to the hypothesis tests are reported: the null hypothesis was rejected for each configuration ($p < 0.05$), confirming the significance of the improvement given by our approach.

5.2. 20NEWS

On these data, we referred to the architectures presented in [32], made by $C16 - FC100$. Also in this case, the comparisons were made varying the number of convolutional filters, and considering the convolutional layer both as static and dynamic. The adopted architecture and parameters configurations are reported in Fig. 3 and in Table 2 respectively.

As it is shown in Fig. 6(a), the introduction of the dynamical layer increases the average performances. In Fig. 6(b) we can observe again how our method has a faster convergence than the static one. In Table 4

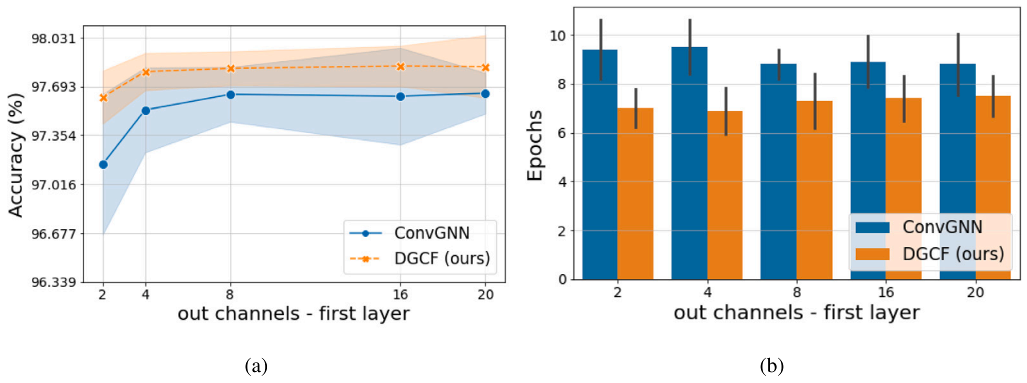


Fig. 4. Results of the experiment on MNIST dataset: (a) mean and standard deviation band of accuracy per configuration; (b) average of the training epochs to convergence per configuration.

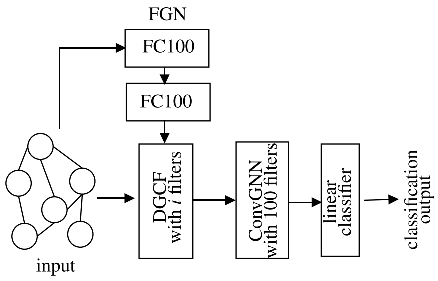


Fig. 5. Architecture adopted for experiments with 20NEWS dataset. See text and Table 2 for further details.

Table 5
Hypothesis tests results for the SEED dataset.

Hypothesis tests					
Configuration	$C2$	$C4$	$C8$	$C16$	$C32$
p-value	0.3118	0.0138	< 0.0001	0.0088	0.0007

the results related to the hypothesis tests are reported: in each case, the null hypothesis was rejected confirming the significance of the improvement given by our method.

5.3. SEED

The base architecture considered for this experiment was similar to the RGNN model proposed by [38], i.e. $Ct - Pooling$. Differently from [38], in this series of experiment domain adaptation [40] techniques were not adopted. Moreover, we chose the global mean pooling across all the nodes on the graph instead of sum pooling, since it gave better results in firsts exploratory experiments. Also in this case, comparisons were made considering the convolutional layer both as static and dynamic, and varying the number of convolutional filters. Finally, the weight decay parameter was introduced to decrease the models' complexity. The adopted architecture and parameters configurations are reported in Fig. 7 and in Table 2.

As it is shown in Fig. 8(a), the introduction of the dynamical layer strongly increases the average performance of the architecture. In Fig. 8(b) it is enhanced the quicker convergence of our method than the static one. In Table 5 the results related to the hypothesis tests are reported: except for the case $C2$, the null hypothesis was rejected for each configuration, confirming that the improvement given by our method is significant.

It is interesting to notice what is shown in Table 6: referring to what is presented in [38], our method overcomes some domain adaptation techniques, such as TCA.

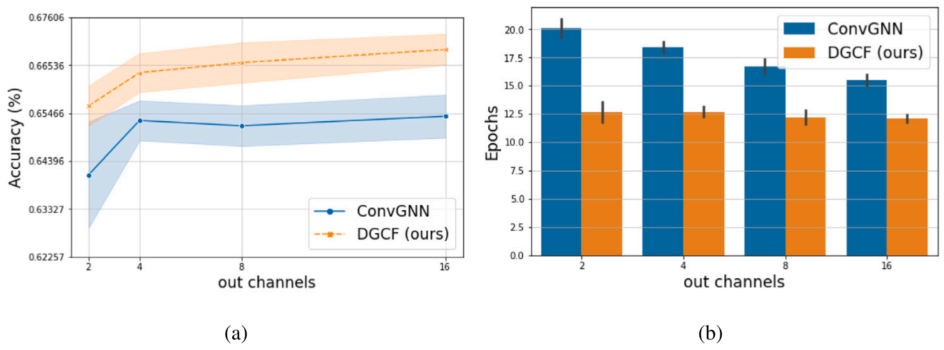


Fig. 6. Results of the experiment on 20NEWS dataset: (a) mean and standard deviation band of accuracy per configuration; (b) average of the training epochs to convergence per configuration.

Table 6
Subject-independent best average classification accuracy (mean ± std) on SEED dataset using different methods, as reported in [38]. In the last row, the best average accuracy of the proposed model was reported. Methods highlighted with * involve the use of Domain Adaptation techniques.

SEED										
Model	SVM	TCA*	SA*	T-SVM	DGCNN	DAN*	BiDANN-S*	BiHDM* (SOTA)	RGNN*	DGCF (ours)
Mean Accuracy ± std.	56.73 ± 16.29	63.64 ± 14.88	69.00 ± 10.89	72.53 ± 14.00	79.95 ± 09.02	83.81 ± 08.56	84.14 ± 06.87	85.40 ± 07.53	85.30 ± 06.72	81.76 ± 05.38

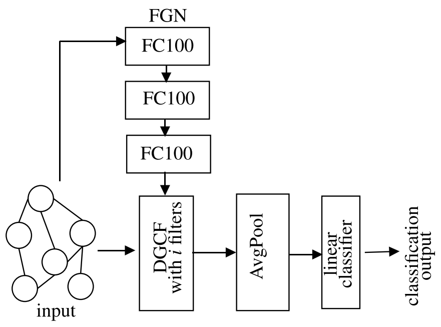


Fig. 7. Architecture adopted for experiments with SEED dataset. See text and Table 2 for further details.

6. Model Analysis on SEED dataset

In this section we propose a visual analysis of the training stages of both the dynamic and static approach on the SEED dataset experiment. The main aim of this analysis consists in inspecting differences in filter generation according to the input. In particular, assuming that common patterns are shared among equally-labelled samples, we expect that filters generated for samples of the same class are similar to each other, while they are different for samples belonging to different classes. We chose the SEED dataset for this analysis since it has the lowest number of classes among the datasets used for the experimental assessment. This analysis was made considering a random subject as test set, and the remaining as training set. Configurations $C2-AvgPool$, for the static model, and $DC2-AvgPool$, for the dynamic model, were considered. The remaining details about the experimental setup follow the Table 2.

The learning processes are graphically described by weights distribution after each training epoch. In Fig. 9, the training of both the static filters are shown. In Fig. 10, the training stage of both of the dynamic filters, produced by the filter-generating network, are shown for each label (negative, neutral and positive, from left to right). Since the filters are generated uniquely for each sample, in order to have a fair evaluation of how they are distributed for each epoch, filters related to correctly classified samples in each epoch are collected and averaged.

From Fig. 10, we can observe how the filter generation process is different for each class: assuming that equally-labelled data should share common patterns, Fig. 10 shows how different ranges of values are covered on the final parameters' configuration for input belonging to different classes, involving that specific filters are obtained according to its features. The direct consequence of this aspect is that feature extraction of the main architecture is enhanced by intrinsic patterns hidden in the sample itself. In facts, we can see that, almost in all cases, satisfying results are achieved using a low number of filters. Moreover, comparing Figs. 9 and 10, we can notice a difference in the ranges of values covered by the last configurations of both the models: static filters have weights included in the range $[-1, 1.5]$, while the dynamic ones have weights overall included in the range $[-0.6, 0.5]$. This aspect involves that the use of the dynamic layer could lead to less complex models, thus avoiding over-fitting/under-fitting [12].

A graphical representation of the filters related to the final configuration of each model was made using heatmaps in Fig. 11, for the static model, and Fig. 12, for the dynamic one. Considered a generic filter $W \in \mathbb{R}^{K\times J}$, the entry W_{ij} represents the numerical value related to the transform of the jth feature of the ith neighbour. For the dynamic case, standard deviations of the weights are represented into the heatmap cells since, as we described above, for this analysis filters were averaged over a selected group of samples.

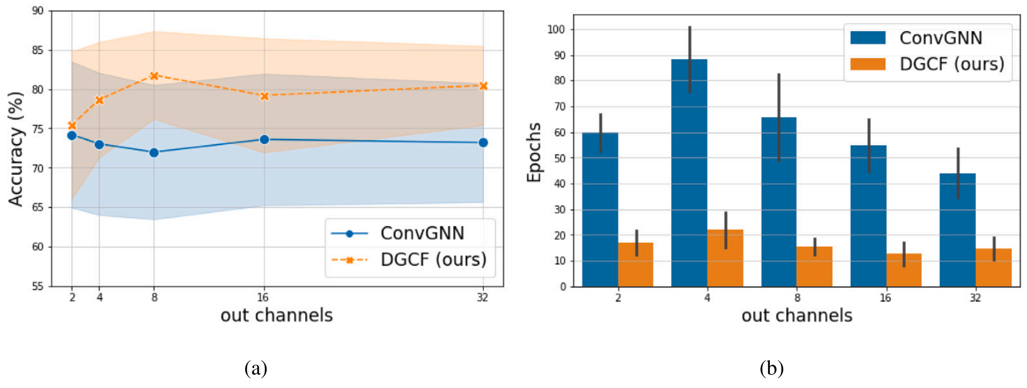


Fig. 8. Results of the experiment on SEED dataset: (a) mean and standard deviation band of accuracy per configuration; (b) average of the training epochs to convergence per configuration.

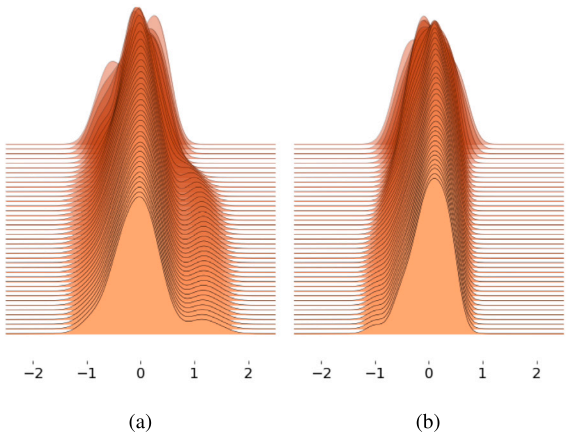


Fig. 9. Ridgeplot weights representation of a static convolutional layer with two filters during the learning process on the SEED dataset. Histograms related to the weights assumed by the first (a) and the second (b) filter are reported for each epoch. Histograms are overlapped from the first (background) to the last epoch (foreground).

In Fig. 12 we can clearly confirm what we observed in Fig. 10: the filter-generating network generates different filters according to the sample label. It is interesting to notice how different features are enhanced in the label related filters: for example, for the positive label, both of the first and the second filter enhance, with high values in absolute value, features 3 and 4 (corresponding to beta and gamma bands), for each neighbour; differently, instead, filters related to the negative label enhances feature 1, corresponding to the theta band.

7. Discussion

The results presented in this study highlight the advantages of using the proposed dynamic approach for GNN. Specifically, introducing DGCF to the architecture increases the average performance with respect to standard convGNN. This improvement is achieved even with simpler architectures requiring fewer convolutional filters. Comparing the results of this study to those reported in [4], the proposed dynamic approach with node-specific filters performs comparably with fewer convolutional filters. This suggests that the dynamic layer can learn informative representations from the graph structure and build suitable convolutional filters. One significant advantage of the proposed dynamic approach is that convergence during the training stage is reached in fewer epochs. This saves computational resources and suggests that the proposed method is better suited for more complex tasks requiring longer training times. Our proposed method performs better on the SEED dataset than many of the compared methods. In particular, our approach outperforms two domain adaptation strategies, that are TCA and SA (see Table 6 and [38]), but performs slightly worse than methods based on adversarial methods. However, note that these approaches attempt to alleviate the *data shift* problem using data coming from the test set. By contrast, our method reaches interesting performances tuning its behaviour with respect to the specific domain and without needing further data.

Finally, the study's results confirm the significance of the improvement given by the proposed approach, as evidenced by the rejection of the null hypothesis for almost all the tested configurations. Additionally, the visual inspection of the learned models reveals that, in nearly all cases, class-specific filters are learned by the model, allowing satisfactory results with fewer filters.

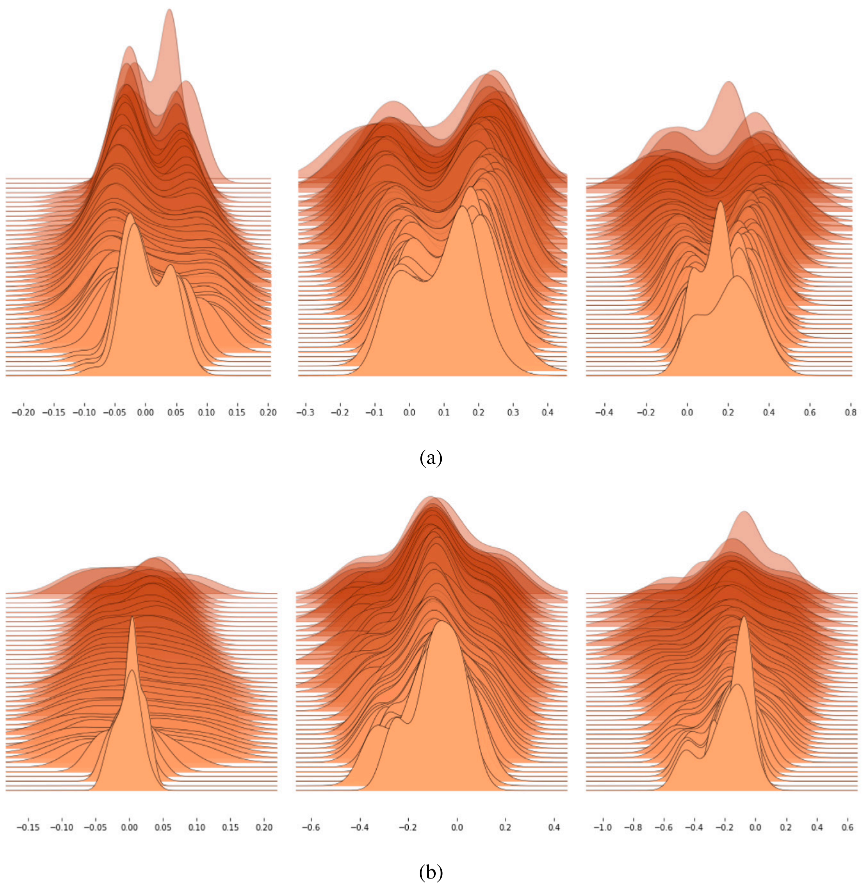
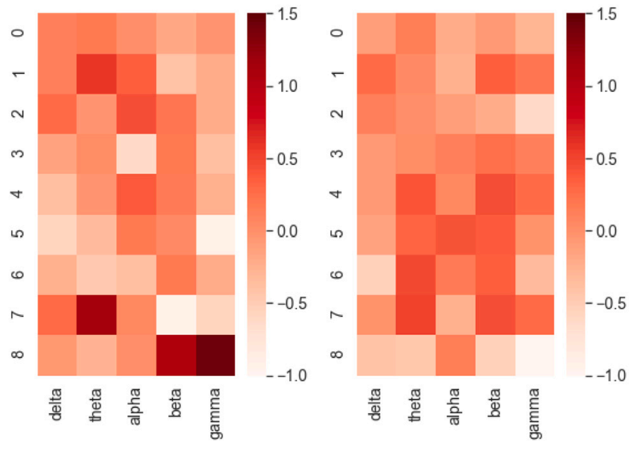


Fig. 10. Ridgeplot weights representation of a DGCF with two filters during the learning process on the SEED dataset. Histograms related to the weights assumed by the first (a) and the second (b) filter are reported for each epoch. Histograms are overlapped from the first (background) to the last epoch (foreground). For each filter, training process related to the negative (left), neutral (centre) and positive (right) labels are represented. The histogram in foreground is related to the final filter configuration.



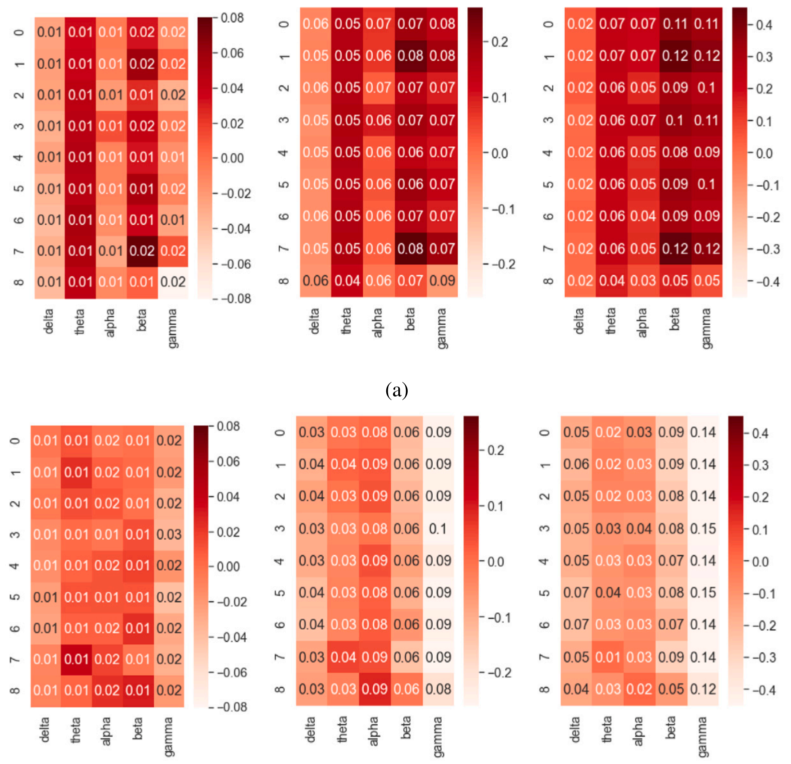
(a) (b)

Fig. 11. Heatmap weights representations of the first (a) and the second (b) filter of a static convolutional layer at the end of the learning process on the SEED dataset. Filters are represented by matrices $W \in \mathbb{R}^{9\times5}$, where 9 is the kernel size and 5 is the number of input features, corresponding to the five EEG bands (see text for further details).

These results suggest that the proposed dynamic approach has important implications for improving the performance of convGNNs, particularly for more complex tasks.

8. Conclusion

In this work, a dynamic method to perform spatial convolution on graph-structured data is proposed. More in detail, combining the idea of having dynamically changeable behaviours in ANNs and convolutional graph neural networks, this work aimed to present a graph convolutional layer capable of performing convolution using node-specific filters to achieve an input-specific filtering operation. We altered the behaviour of a convolutional layer in a dynamic fashion using a filter-generating network. In this way, the proposed graph convolutional layer learns and applies input-specific filters, customizing the filtering operation according to its input graph. We ran a series of experiments to assess the improvements in using a dynamic approach to generate convolutional filters. It empirically emerged that our proposed strategy leads to better performances than those achieved using the static convolution on graphs. As we observed from Figs. 10 and 12, FGNs learn to produce class-specific filters, making the convolution operation input-specific actually. Furthermore, convergence is reached in fewer epochs, reducing training time in a significant matter. Finally, using regularization techniques, the use of an external module leads to filters having smaller weights than the static filters, resulting in an overall lower complexity of the main architecture. In conclusion, this study showed that the proposed approach has numerous advantages over standard convGNNs. By introducing the proposed dynamical layer to the architecture, the average performance of the model can increase, even with simpler architectures requiring fewer convolutional filters. The proposed dynamic layer can learn informative representations from the graph structure and build suitable convolutional filters, leading



(a)

(b)

Fig. 12. Heatmap weights representations of the first (a) and the second (b) filter of DGCF at the end of the learning process on the SEED dataset, averaged over the correctly classified samples. For each filter, averages related to the negative (left), neutral (centre), and positive (right) labels are shown. Filters are represented by matrices $W \in \mathbb{R}^{9\times5}$, where 9 is the kernel size and 5 is the number of input features, corresponding to the five EEG bands (see text for further details).

to comparable or even better results with fewer parameters to learn compared to other methods. One significant advantage is that the proposed approach reaches convergence during the training stage in fewer epochs, making it better suited for more complex tasks. Moreover, the proposed approach outperforms standard strategies, including some Domain Adaptation strategies, without requiring data from the target domain during the learning stage. The improvement given by the proposed approach is highlighted by the rejection of the null hypothesis for almost all the tested configurations.These aspects were evident in the emotion recognition from EEG signals, which is a complex task to achieve. In future work, we would like to introduce and analyse the use of a dynamic local filtering layer having local filters generated for each neighbourhood, as proposed by [11] in the image domain. Currently, since the filter-generating network takes as input the entire graph, our strategy is constrained to data having a fixed graph topology. Using a local dynamic convolution, we could overcome this limit by extending this layer's functionality to data with non-fixed graphs topologies.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

All the data are publicly available. Source code are avaialable on GitHub.

Acknowledgements

This work is supported by the European Union - FSE-REACT-EU, PON Research and Innovation 2014–2020 DM1062/2021 contract number 18-I-15350-2, and was partially supported by the Ministry of University and Research, PRIN research project "BRIO – BIAS, RISK, OPACITY in AI: design, verification and development of Trustworthy AI.", Project no. 2020SSKZ7R, and by the Ministry of Economic Development, "INtegrated Technologies and ENhanced SEnsing for cognition and rehabilitation" (INTENSE) project. Furthermore, we acknowledge financial support from the PNRR MUR project PE0000013-FAIR (CUP: E63C22002150007).

References

[1] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai, et al., Recent advances in convolutional neural networks, Pattern Recognit. 77 (2018) 354–377.
[2] F. Scarselli, M. Gori, A.C. Tsoi, M. Hagenbuchner, G. Monfardini, The graph neural network model, IEEE Trans. Neural Netw. 20 (1) (2008) 61–80.
[3] W. Peng, X. Hong, G. Zhao, Tripool: Graph triplet pooling for 3D skeleton-based action recognition, Pattern Recognit. 115 (2021) 107921.
[4] Y. Hechtlinger, P. Chakravarti, J. Qin, A generalization of convolutional neural networks to graph-structured data, 2017, arXiv preprint arXiv:1704.08165.
[5] F. Donnarumma, R. Prevete, G. Trautteur, Programming in the brain: A neural network theoretical framework, Connect. Sci. 24 (2–3) (2012) 71–90.
[6] D. Ha, A.M. Dai, Q.V. Le, HyperNetworks, in: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, 2017.
[7] M. Simonovsky, N. Komodakis, Dynamic edge-conditioned filters in convolutional neural networks on graphs, in: 2017 IEEE Conference on Computer Vision and Pattern Recognition, (CVPR), 2017, pp. 29–38.
[8] E. Nachmani, L. Wolf, Hyper-graph-network decoders for block codes, Adv. Neural Inf. Process. Syst. 32 (2019).
[9] C. Zhang, M. Ren, R. Urtasun, Graph HyperNetworks for neural architecture search, in: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, la, USA, May 6-9, 2019.
[10] I. Balažević, C. Allen, T.M. Hospedales, Hypernetwork knowledge graph embeddings, in: International Conference on Artificial Neural Networks, Springer, 2019, pp. 553–565.
[11] X. Jia, B. De Brabandere, T. Tuytelaars, L.V. Gool, Dynamic filter networks, Adv. Neural Inf. Process. Syst. 29 (2016) 667–675.
[12] C. Bishop, Bishop-pattern recognition and machine learning-Springer 2006, Antimicrob. Agents Chemother. (2014) 03728-03714.
[13] Z.L. Li, G.W. Zhang, J. Yu, L.Y. Xu, Dynamic graph structure learning for multivariate time series forecasting, Pattern Recognit. 138 (2023) 109423.
[14] X.H. Li, F. Yin, H.S. Dai, C.L. Liu, Table structure recognition and form parsing by end-to-end object detection and relation parsing, Pattern Recognit. 132 (2022) 108946.
[15] J. Wang, W. Wang, L. Wang, Z. Wang, D.D. Feng, T. Tan, Learning visual relationship and context-aware attention for image captioning, Pattern Recognit. 98 (2020) 107075.
[16] Y. Chen, G. Ma, C. Yuan, B. Li, H. Zhang, F. Wang, W. Hu, Graph convolutional network with structure pooling and joint-wise channel attention for action recognition, Pattern Recognit. 103 (2020) 107321.
[17] L. Tian, P. Wang, G. Liang, C. Shen, An adversarial human pose estimation network injected with graph structure, Pattern Recognit. 115 (2021) 107863.
[18] B. Fasel, J. Luettin, Automatic facial expression analysis: A survey, Pattern Recognit. 36 (1) (2003) 259–275.
[19] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, S.Y. Philip, A comprehensive survey on graph neural networks, IEEE Trans. Neural Netw. Learn. Syst. (2020).
[20] R. Li, S. Wang, F. Zhu, J. Huang, Adaptive graph convolutional neural networks, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 32, (1) 2018.
[21] J. Atwood, D. Towsley, Diffusion-convolutional neural networks, in: Advances in Neural Information Processing Systems, 2016, pp. 1993–2001.
[22] S. Zhang, H. Tong, J. Xu, R. Maciejewski, Graph convolutional networks: A comprehensive review, Comput. Soc. Netw. 6 (1) (2019) 11.
[23] J. Bruna, W. Zaremba, A. Szlam, Y. Lecun, Spectral networks and locally connected networks on graphs, in: International Conference on Learning Representations, (ICLR2014), CBLS, April 2014, 2014.
[24] M. Henaff, J. Bruna, Y. LeCun, Deep convolutional networks on graph-structured data, 2015, arXiv preprint arXiv:1506.05163.
[25] A. Micheli, Neural network for graphs: A contextual constructive approach, IEEE Trans. Neural Netw. 20 (3) (2009) 498–511.
[26] M. Defferrard, X. Bresson, P. Vandergheynst, Convolutional neural networks on graphs with fast localized spectral filtering, Advances Neural Inf. Process. Syst. 29 (2016) 3844–3852.
[27] T.N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, in: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, 2017.
[28] M. Niepert, M. Ahmed, K. Kutzkov, Learning convolutional neural networks for graphs, in: International Conference on Machine Learning, 2016, pp. 2014–2023.
[29] S. Fu, W. Liu, Y. Zhou, L. Nie, HpLapGCN: Hypergraph p-Laplacian graph convolutional networks, Neurocomputing 362 (2019) 166–174.
[30] S. Bai, F. Zhang, P.H. Torr, Hypergraph convolution and hypergraph attention, Pattern Recognit. 110 (2021) 107637.
[31] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, K. Weinberger, Simplifying graph convolutional networks, in: International Conference on Machine Learning, PMLR, 2019, pp. 6861–6871.
[32] Q. Zhang, J. Chang, G. Meng, S. Xu, S. Xiang, C. Pan, Learning graph structure via graph convolutional networks, Pattern Recognit. 95 (2019) 308–318.
[33] D.C. Thang, H.T. Dat, N.T. Tam, J. Jo, N.Q.V. Hung, K. Aberer, Nature vs. nurture: feature vs. structure for graph neural networks, Pattern Recognit. Lett. 159 (2022) 46–53.
[34] J. Xue, B. Zhang, Q. Qiang, Local linear embedding with adaptive neighbors, Pattern Recognit. 136 (2023) 109205.
[35] X. Wang, J. Zhu, Z. Xu, K. Ren, X. Liu, F. Wang, Local nonlinear dimensionality reduction via preserving the geometric structure of data, Pattern Recognit. (2023) 109663.
[36] W.L. Zheng, B.L. Lu, Investigating critical frequency bands and channels for EEG-based emotion recognition with deep neural networks, IEEE Trans. Autonom. Ment. Dev. 7 (3) (2015) 162–175.
[37] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space, in: 1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, 2013, 2013.
[38] P. Zhong, D. Wang, C. Miao, EEG-based emotion recognition using regularized graph neural networks, IEEE Trans. Affect. Comput. (2020).
[39] J. Snoek, H. Larochelle, R.P. Adams, Practical bayesian optimization of machine learning algorithms, Advances Neural Inf. Process. Syst. 25 (2012).
[40] M. Wang, W. Deng, Deep visual domain adaptation: A survey, Neurocomputing 312 (2018) 135–153.

Andrea Apicella received the M.Sc. degree in Computer Science and the Ph.D. degree in Mathematics and Computer Science from Federico II University of Naples, Italy, in 2014 and 2019, respectively. He is currently a research fellow in the Department of Information Technology and Electrical Engineering of University of Naples Federico II and he is a research associate of the ARHeMLab (Augmented Reality for Health Monitoring Laboratory) and AIPA Lab (Laboratory of Artificial Intelligence, Privacy & Applications) sited in Naples. The current research topics of Andrea Apicella cover EEG signal processing for emotion recognition and attention monitoring using

Artificial Intelligence methods, and eXplainable Artificial Intelligence (XAI) approaches for explaining the AI system's decisions.

Francesco Isgrò was awarded a Master degree in Mathematics from Università di Palermo (1994), and a Ph.D. in Computer Science from Heriot-Watt University in Edinburgh (UK) in 2001. He worked as Research Assistant at Heriot-Watt University and Università di Genova (Italy), and since 2006 is permanent staff member at Università di Napoli Federico II (Italy), where he teaches the courses of Computer Vision, and Programming. His research interests cover various areas of image processing, computer vision and machine learning methods. He has co-authored more than 80 scientific papers. He has served on the technical and organizing committees of several conferences, and has refereed papers for various journals.

Andrea Pollastro graduated with a M.Sc. degree in Computer Science from the University of Naples Federico II in 2019. He is currently a Ph.D. Student in the Department of Information Technology And Electrical Engineering of the University of Naples Federico II since 2020. His research interests include machine learning and neural networks. Currently, he is working at the Advanced Light Source (ALS), the Lawrence Berkeley National Laboratory (LBNL) third-generation synchrotron light source, on application of Machine Learning methods to beam-size control and power-supply anomaly detection in storage rings in collaboration with the Accelerator Physics Group (APG).

Roberto Prevete (M.Sc. in Physics, Ph.D. in Mathematics and Computer Science) is an Assistant Professor of Computer Science at the Department of Electrical Engineering and Information Technologies (DIETI), University of Naples Federico II, Italy. His current research interests include computational models of brain mechanisms, machine learning and artificial neural networks and their applications. His research has been published in international journals such as Biological Cybernetics, Experimental Brain Research, Neurocomputing, Neural Networks and Behavioral and Brain Sciences.