Deformable Graph Convolutional Networks

Jinyoung Park, Sungdong Yoo, Jihwan Park, Hyunwoo J. Kim*

Department of Computer Science and Engineering, Korea University
{lpmn678, ysd424, jseven7071, hyunwoojkim}@korea.ac.kr

Abstract

Graph neural networks (GNNs) have significantly improved the representation power for graph-structured data. Despite of the recent success of GNNs, the graph convolution in most GNNs have two limitations. Since the graph convolution is performed in a small local neighborhood on the input graph, it is inherently incapable to capture long-range dependencies between distance nodes. In addition, when a node has neighbors that belong to different classes, *i.e.*, *heterophily*, the aggregated messages from them often negatively affect representation learning. To address the two common problems of graph convolution, in this paper, we propose Deformable Graph Convolutional Networks (Deformable GCNs) that adaptively perform convolution in multiple *latent spaces* and capture short/long-range dependencies between nodes. Separated from node representations (features), our framework simultaneously learns the *node positional embeddings* (coordinates) to determine the relations between nodes in an end-to-end fashion. Depending on node position, the convolution kernels are deformed by deformation vectors and apply different transformations to its neighbor nodes. Our extensive experiments demonstrate that Deformable GCNs flexibly handles the heterophily and achieve the best performance in node classification tasks on six heterophilic graph datasets.

1 Introduction

Graphs are flexible representations for modeling relations in data analysis problems and are widely used in various domains such as social network analysis (Wang, Cui, and Zhu 2016), recommender system (Berg, Kipf, and Welling 2017), chemistry (Gilmer et al. 2017), natural language processing (Erkan and Radev 2004), and computer vision (Johnson et al. 2015). In recent years, Graph Neural Networks (GNNs) have achieved great success in many graph-related applications such as node classification (Kipf and Welling 2017; Hamilton, Ying, and Leskovec 2018), link prediction (Zhang and Chen 2018; Schlichtkrull et al. 2018), and graph classification (Errica et al. 2019; Ying et al. 2018). Most existing GNNs learn node representations via message passing schemes, which iteratively learn the hidden representation of each node by aggregating messages from its local neighborhoods. For example, Graph Convolution Networks (GCNs) (Kipf and Welling 2017) operate convolutions on input graphs inspired by first-order approximation of spectral graph convolutions (Hammond, Vandergheynst, and Gribonval 2011).

However, most graph convolution that aggregates messages from local neighborhoods implicitly assumes that input graphs are homophilic graphs, where connected nodes have similar features or belong to the same class. So the smoothing over the input graphs effectively removes noise in the input features and significantly improve the representational power when the assumption holds. However, on heterophilic graphs where connected nodes have dissimilar features and different labels, the conventional graph convolutional neural networks often underperform simple methods such as a multi-layer perceptron (MLP) that completely ignores the graph structure. In addition, since the conventional graph convolution receives messages from local neighbors, it has the limited capability to capture long-range dependencies between distant yet relevant nodes for the target tasks.

To address these limitations, we propose a Deformable Graph Convolutional Network (Deformable GCN) that softly changes the receptive field of each node by adaptively aggregating the outputs of deformable graph convolution in multiple latent spaces. Started from a general definition of the discrete convolution with finite support, we extend the deformable 2D convolution (Dai et al. 2017) to a latent space for graph-structured data. Similar to the convolution defined on a grid space for images, our convolution kernel generates different transformations for various relations. Our framework models useful relations between nodes represented by the difference of learned *node positional embeddings*. Our **contributions** are as follows:

- We propose a Deformable Graph Convolution (Deformable GConv) that performs convolution in a latent space and adaptively deforms the convolution kernels to handle heterophily and variable-range dependencies between nodes.
- We propose novel architecture Deformable Graph Convolution Networks (Deforamble GCN) that simultaneously learn node representations (features) and node positional embeddings (coordiantes) and efficiently perform Deformable GConv in multiple latent spaces using

*is the corresponding author.

latent neighborhood graphs.

- Our experiments demonstrate the effectiveness of Deformable GCN in the node classification task on homophily and heterophily graphs. Also, the *interpretable* attention scores in our framework provide insights which relation (or latent space) is beneficial to the target task.

2 Related Works

Graph Neural Networks. Graph neural networks have been studied for representing graph-structured data in recent years. Based on spectral graph theory, ChebyNet (Defferrard, Bresson, and Vandergheynst 2016) designed fast localized convolutional filters and reduced computational cost. Motivated by it, (Kipf and Welling 2017) proposed GCN, which is simplified convolutional networks based on the first-order approximation of the spectral convolutions. There are several studies to improve the performance by message passing processes (Hamilton, Ying, and Leskovec 2018; Xu et al. 2018) and attention-based models (Veličković et al. 2017; Yun et al. 2019).

However, the studies introduced above have been developed based on the *homophily assumption* that connected nodes have similar characteristics. This assumption often leads to performance degradation on *heterophilic graphs*. Moreover, most existing GNNs cannot capture the *long-range dependencies* between distant nodes because they aggregate messages only in a local neighborhood. Recently, there are several studies have attempted to address the problems (Bo et al. 2021; Liu, Wang, and Ji 2020; Zhu et al. 2020, 2021). Geom-GCN (Pei et al. 2020) proposed a novel geometric aggregation scheme in a latent space that captures long-range dependencies through structural information. However, Geom-GCN has some limitations that they define convolution on a grid that is a manually divided latent space and require pre-trained embedding methods in the geometric aggregation step. Unlike Geom-GCN, our models apply deformable convolution kernel on the latent space with the relation vectors defined in a continuous latent space and utilize learnable multiple embeddings to softly grant relations in an end-to-end fashion.

Deformable Convolution. Convolutional neural networks (CNNs) have achieved great success in various fields (He et al. 2016, 2017). However, the convolution kernels are limited to model large and unknown transformations since they are defined in a fixed structure. To address these limitations, (Dai et al. 2017; Zhu et al. 2019) proposed deformable convolution networks that adaptively change the shape of convolution kernels by learning offsets for deformation Because of the feasibility and effectiveness, deformable convolution networks are used in various fields, such as point cloud (Thomas et al. 2019), image generation (Siarohin et al. 2018), and video tasks (Wang et al. 2019). Inspired by this line of work, we study a deformable graph convolution to adaptively handle diverse relations between an ego-node and its neighbors.

3 Method

The goal of our framework is to address the limitations of existing graph convolutions that learn node representations in a small neighborhood with the homophily assumption. In other words, existing GNNs often poorly perform when neighbors belong to different classes and have dissimilar features. Also, most GNNs with a small number of layers cannot model the long-range dependency between distant nodes. To address the limitations, we propose Deformable Graph Convolutional Networks that perform deformable convolution on latent graphs. Our framework softly changes the shape (or size) of the receptive field for each node. This allows our framework to adaptively handle homophilic/heterophilic graphs as well as short/long-range dependency. Before introducing our frameworks, we first briefly summarize notations and the basic concepts of graph neural networks.

3.1 Preliminaries

Notations. Let $G = (V, E)$ denote an input graph with a set of nodes V and a set of edges $E \subseteq V \times V$. Each node $v \in V$ has a feature vector $\mathbf{x}_v \in \mathbb{R}^{d_x}$ and the edge between node u and node v is represented by $(u, v) \in E$. The neighborhoods of node v on input graph is denoted by $N(v) = \{u \in V | (u, v) \in E\}$.

Graph Neural Networks. To learn representation for graph-structured data, most existing GNNs perform message-passing frameworks as the following equation (Gilmer et al. 2017; Xu et al. 2018):

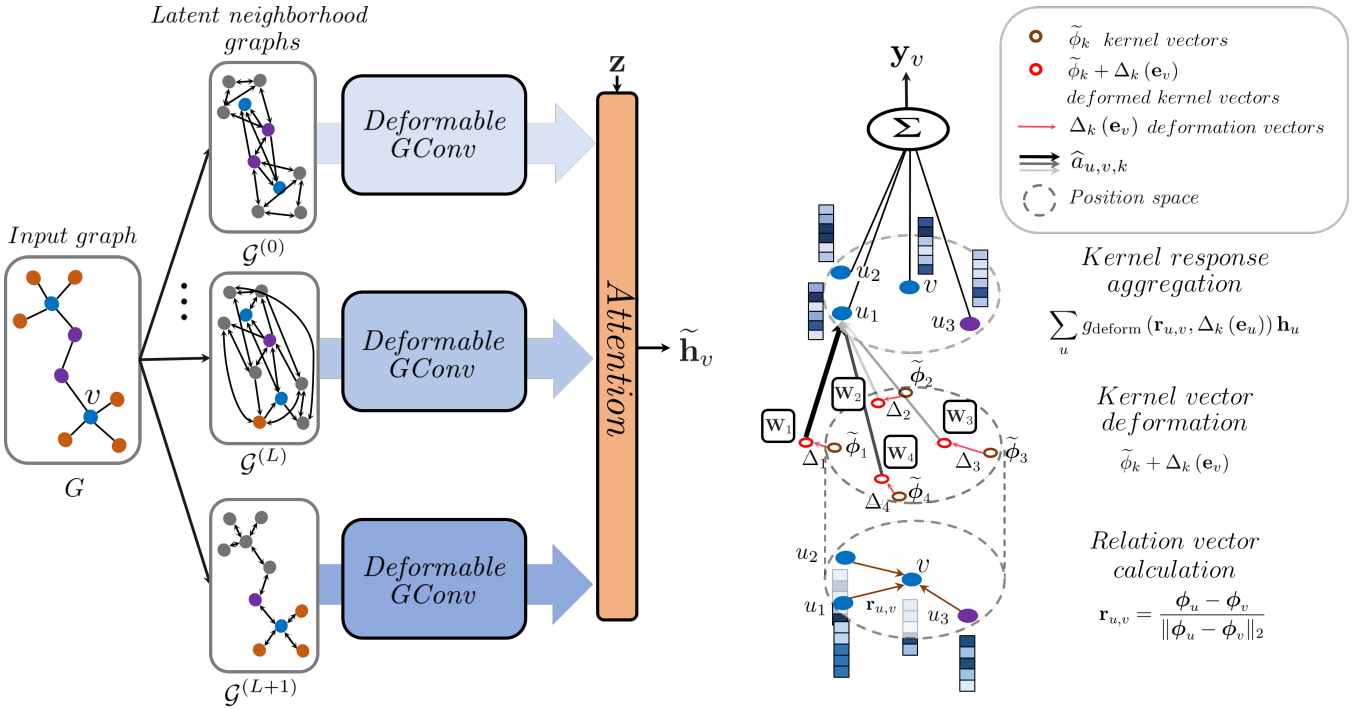
$$\mathbf{h}_v^{(l)} = \sigma\left(\mathbf{W}^{(l)} \cdot \text{AGGREGATE}\left(\mathbf{h}_u^{(l-1)} : u \in \widetilde{N}(v)\right)\right), \tag{1}$$

where $\mathbf{h}_v^{(0)} = \mathbf{x}_v$, $\mathbf{h}_v^{(l)} \in \mathbb{R}^{d_{h^{(l)}}}$ is a hidden representation of node v in a l-th layer, $\widetilde{N}(v) = \{v\} \cup \{u \in V | (u, v) \in E\}$ indicates neighbors of node v with a self-loop, $\mathbf{W}^{(l)}$ is a learnable weight matrix at the l-th layer, AGGREGATE is an aggregation function characterized by the particular model, and σ is a non-linear activation function. $\widetilde{N}(v)$ determines the receptive field of the graph convolution and it is usually a one-hop ego-graph. For example, GCN (Kipf and Welling 2017) is a specific instance of (1). GCN can be written as

$$\mathbf{h}_v^{(l)} = \sigma\left(\sum_{u \in \widetilde{N}(v)} \left(\deg(v)\deg(u)\right)^{-1/2} \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)}\right), \tag{2}$$

where $\deg(v)$ is the degree of node v.

Even GNNs such as GCN (Kipf and Welling 2017) and GAT (Veličković et al. 2017) have been successfully applied to various graph-based tasks, their success is limited to *homophilic* graphs (Pei et al. 2020; Zhu et al. 2020), which are graphs that linked nodes often have similar properties (McPherson, Smith-Lovin, and Cook 2001). Recent works (Li, Han, and Wu 2018; Wu et al. 2019) showed that GCN makes node embeddings smoother within their peripherals since GCN can be a specific form of Laplacian smoothing (Li, Han, and Wu 2018). For this reason, graph neural



(a) Deformable GCN (b) Deformable GConv

Figure 1: Overall structure of Deformable GCN and Deformable GConv. (a) In Deformable GCN, at the beginning of training, latent neighborhood graphs, $\{\mathcal{G}^{(l)}\}_{l=0}^{L+1}$, are constructed to define neighborhoods for the Deformable Graph Convolution (Deformable GConv). Then, Deformable GConv is applied on each latent neighborhood graph and the outputs of the convolution $\{\mathbf{y}_v^{(l)}\}_l$ are adaptively aggregated for representing $\widetilde{\mathbf{h}}_v$ using an attention mechanism. (b) Our Deformable GConv performs graph convolution in a latent (position) space. For more flexible graph convolution, Deformable GConv adaptively deforms convolution kernels $g_{\text{deform}}(\cdot,\cdot)$ for each center node v by kernel vector deformation $\Delta_k(\mathbf{e}_v)$.

networks have difficulty adapting to graphs that linked nodes often have different properties, called *heterophilic* graphs.

In our work, we consider both heterophilic and homophilic graphs, unlike standard graph neural networks that have mainly focused on homophilic graphs. To have enough representation power on heterophilic graphs, we generate latent graphs for linking distant nodes with similar property according to their latent embeddings.

3.2 Deformable Graph Convolution

We here introduce a Deformable Graph Convolution (Deformable GConv), which softly changes the receptive fields and adaptively aggregates messages from neighbors on the multiple latent graphs. The overall structure of Deformable GConv is illustrated in Figure 1. Our Deformable Graph Convolution can be derived from a general definition of the discrete convolution by a kernel with finite support. The convolution of a feature map $\mathcal{H}$ by a kernel g at a point v is given as:

$$\mathbf{y}_v = (\mathcal{H} * g)(v) = \sum_{u \in \widetilde{\mathcal{N}}(v)} g(\mathbf{r}_{u,v})\,\mathbf{h}_u, \qquad (3)$$

where $\mathbf{y}_v \in \mathbb{R}^{d_y}$ is the output of the convolution, $\mathbf{h}_u \in \mathbb{R}^d$ is a feature at u, and relation vector $\mathbf{r}_{u,v}$ represent the relation between u and v, $\widetilde{\mathcal{N}}(v)$ is the neighborhood of v that coincides with the finite support of g centered at v. $g(\mathbf{r}_{u,v})$ is a linear function to transform $\mathbf{h}_u$ and it varies depending on the relation vector. For example, in a 2D convolution with a 3×3 kernel, $\mathbf{r}_{u,v} = \boldsymbol{\phi}_u - \boldsymbol{\phi}_v$, where $\boldsymbol{\phi}_u$, $\boldsymbol{\phi}_v$ are the coordinates of u and v. For each relative position, a linear function $g(\mathbf{r}_{u,v}) \in \mathbb{R}^{d_y \times d_h}$ is applied.

In the graph domain, a GCN layer defined in (2) (without the activation function) can be viewed as a specific instantiation of (3) with $g(\mathbf{r}_{u,v}) = (\deg(v)\deg(u))^{-1/2}\mathbf{W}$. So, the GCN relations are determined by the degree of node u and v. Also, except for the normalization, GCN performs the same linear transformation for all the relations different from standard 2D convolution.

Our framework extends the graph convolution to more flexible and deformable graph convolutions defining a relation vector $\mathbf{r}_{u,v}$, a kernel function $g(\cdot)$, and its support (or a neighborhood). To extend the relation of nodes beyond the adjacency on the input graph G, we first embed nodes in a position space, which is a latent space to determine the coordinates of nodes using a *node positional embedding*. Then, we compute the *relation vector* of the nodes by a function of the node positional embeddings.

Node positional embedding. In our framework, each node is embedded in a latent space called *position space* using a node embedding method. Since the node embeddings are used only for the coordinates of nodes and their relations, we name this *node positional embedding*. For each node, our framework learns both node positional embeddings (coordinates) and node representations (features).

Node positional embedding is computed by a simple procedure. Given a node v, its node positional embedding $\boldsymbol{\phi}_v^{(l)}$ is the projected input features after smoothing l times on the original input graph G. It can be written as

$$\begin{aligned} &\boldsymbol{\phi}_v^{(l)} = \mathbf{W}_{\phi}^{(l)} \mathbf{e}_v^{(l)}, \text{ where} \\ &\mathbf{e}_v^{(0)} = \mathbf{x}_v, \quad \mathbf{e}_v^{(l)} = \frac{1}{\widetilde{\deg}(v)} \sum_{u \in \widetilde{N}(v)} \mathbf{e}_u^{(l-1)}, \end{aligned} \tag{4}$$

where $\mathbf{W}_{\phi}^{(l)}$ is a learnable project matrix, $\widetilde{\deg}(v)$ indicates a degree of node v with self-loop and $\mathbf{e}_v^l$ is the l-time smoothed input features of node v. Each node has $L+1$ node positional embeddings. Alternatively, for node positional embedding $\boldsymbol{\phi}_v$, other node embedding methods can be used such as LINE (Tang et al. 2015), Node2Vec (Grover and Leskovec 2016), distance encoding (Li et al. 2020), Poincare embedding in hyperbolic geometry (Nickel and Kiela 2017), and other various node embedding methods. But our preliminary experiment showed that our simple node positional embedding was sufficient for our model. So in this paper, we did not use any external node embedding method.

Relation vector. The relation between nodes is represented by a relation vector $\mathbf{r}_{u,v}$. One natural choice is the relative position of neighbor node u from node v in the position space. In our framework, we use normalized relation vectors with an extra dimension to encode the relation of nodes with identical positional embeddings. The relation vector for a neighbor node u of node v with node positional vectors $\boldsymbol{\phi}_u, \boldsymbol{\phi}_v \in \mathbb{R}^{d_\phi}$ is given as

$$\mathbf{r}_{u,v} = \begin{cases} \left[\mathbf{r}'_{u,v} || 0\right] \in \mathbb{R}^{d_{\phi}+1}, & \text{if } \boldsymbol{\phi}_u \neq \boldsymbol{\phi}_v \\ [0, 0, \cdots, 1] \in \mathbb{R}^{d_{\phi}+1}, & \text{otherwise} \end{cases} \tag{5}$$

where $\mathbf{r}'_{u,v} = \frac{\boldsymbol{\phi}_u - \boldsymbol{\phi}_v}{\|\boldsymbol{\phi}_u - \boldsymbol{\phi}_v\|_2}$ and $[||]$ is concatenation operator.

Kernel function. As discussed with (3), a kernel function g yields linear functions to transform hidden representations of neighbor nodes. Our kernel function g on $\mathbf{r}_{u,v}$ is defined as:

$$\begin{aligned} g\left(\mathbf{r}_{u,v}\right) &= \sum_{k=1}^{K} a_{u,v,k} \mathbf{W}_k, \\ \text{where } a_{u,v,k} &= \exp\left(\mathbf{r}_{u,v}^{\top} \widetilde{\boldsymbol{\phi}}_k\right) / Z. \end{aligned} \tag{6}$$

$\widetilde{\boldsymbol{\phi}}_k \in \mathbb{R}^{d_\phi+1}$ is a kernel vector and $\mathbf{W}_k \in \mathbb{R}^{d_y \times d_h}$ is a corresponding transformation matrix. Both are learnable parameters. $Z \in \mathbb{R}$ is a normalization factor. The function value of g, which is a linear transformation, varies depending on the relation vector $\mathbf{r}_{u,v}$ but for the same relation, the same linear transformation is returned when Z is a constant. This is the same as a standard 2D convolution that applies the identical linear transformation at the same relative position from the center of the kernel. One interesting difference is that since a standard 2D convolution kernel has a linear transformation matrix for each relative position, the number of its transformation matrices increases as the kernel size increases whereas our kernel g differentially combines a fixed number of $\{\mathbf{W}_k\}_{k=1}^K$ matrices depending on the relation vector $\mathbf{r}_{u,v}$. Thereby the number of parameters of our convolution does not depends on the kernel size anymore.

Deformable Graph Convolution. The kernel function can be further extended for a more flexible and deformable convolution. We first normalize the weight $a_{u,v,k}$ for a transformation matrix $\mathbf{W}_k$ by $Z = \sum_{u'} \exp\left(\mathbf{r}_{u',v}^{\top} \widetilde{\boldsymbol{\phi}}_k\right)$. Then the kernel becomes adaptive. This is similar to the dot-product attention in (Vaswani et al. 2017). We found that the dot-product attention can be viewed as a variant of convolution. For more details, see the supplement. Now the kernel yields a transformation matrix for each neighbor considering not only the relation between a center node v and a neighbor node u but also the relations between neighbor nodes. In addition, the kernel point $\widetilde{\boldsymbol{\phi}}_k$ is dynamically translated by deformation vector $\Delta_k(\mathbf{e}_v) \in \mathbb{R}^{d_\phi+1}$ depending on the smoothed input features $\mathbf{e}_v \in \mathbb{R}^{d_x}$ of the center node v. Putting these pieces together, we define our Deformable Graph Convolution as:

$$\mathbf{y}_v = \sum_{u \in \widetilde{\mathcal{N}}(v)} g_{\text{deform}}\left(\mathbf{r}_{u,v}, \Delta_k\left(\mathbf{e}_v\right)\right) \mathbf{h}_u, \tag{7}$$

where $g_{\text{deform}}\left(\mathbf{r}_{u,v}, \Delta_k\left(\mathbf{e}_v\right)\right) = \sum_{k=1}^{K} \widehat{a}_{u,v,k} \mathbf{W}_k$ and $\widehat{a}_{u,v,k} = \frac{\exp\left(\mathbf{r}_{u,v}^{\top}\left(\widetilde{\boldsymbol{\phi}}_k + \Delta_k(\mathbf{e}_v)\right)\right)}{\sum_{u'} \exp\left(\mathbf{r}_{u',v}^{\top}\left(\widetilde{\boldsymbol{\phi}}_k + \Delta_k(\mathbf{e}_v)\right)\right)}$. In our experiments, the deformation vector is generated by a simple MLP network with one hidden layer.

Latent Neighborhood Graph. The Deformable Graph Convolution is computed within a neighborhood $\widetilde{\mathcal{N}}(v)$. To efficiently compute the neighborhoods, k-nearest neighbors for each node are computed with respect to ℓ_2 distance of smoothed input features, *i.e.*, $\|\mathbf{e}_u - \mathbf{e}_v\|_2$. Since in our framework the input feature smoothing does not have any learnable parameters, the smoothed features $\{\mathbf{e}_u\}$ do not change during training. Thereby, the kNN graph generation is performed once at the beginning of training. Due to the kNN graphs, our Deformable Graph Convolution can be viewed as a graph convolution on the kNN graphs. Seemingly, the neighborhood computed by node positional embedding, $\boldsymbol{\phi}_v$, is more plausible following (Dai et al. 2017). But, it requires huge computational costs for pairwise distance computation of all nodes at every iteration. Also, in practice, the drastic changes of neighborhoods caused by positional embedding learning often lead to unstable numerical optimization. So, in this work, we use the smoothed input features $\{\mathbf{e}_v\}_v$ for the neighborhood computation.

3.3 Deformable Graph Convolutional Networks

With our Deformable Graph Convolution, we design our Deformable Graph Convolution Network (Deformable GCN). The overall structure is depicted in Figure 1. Deformable GCN utilizes multiple node positional embeddings. In other words, it generates multiple neighborhood graphs. Let $\mathbf{e}_v^{(l)}$ denote the input features at node v that are smoothed l times on the original graph G. A Latent Neighborhood Graph, *i.e.*, kNN graph, constructed based on $\{\mathbf{e}_v^{(l)}\}_{v\in V}$ is denoted by $\mathcal{G}^{(l)}$. Deformable GCN generates L+1 kNN graphs $\{\mathcal{G}^{(l)}\}_{l=0}^{l=L}$. Combining with the original input graph $\mathcal{G}^{(L+1)} = G$, Deformable GCN performs the Derformable GConv on each neighborhood graph. The outputs of convolution on $\mathcal{G}^{(l)}$, denoted by $\mathbf{y}_v^{(l)}$, are adaptively aggregated on each node by a simple attention mechanism as:

$$\widetilde{\mathbf{h}}_v = \sum_{l=0}^{L+1} s_v^{(l)} \widetilde{\mathbf{y}}_v^{(l)}, \quad s_v^{(l)} = \frac{\exp\left(\mathbf{z}^\top \widetilde{\mathbf{y}}_v^{(l)}\right)}{\sum_{l'=0}^{L+1} \exp\left(\mathbf{z}^\top \widetilde{\mathbf{y}}_v^{(l')}\right)}, \quad (8)$$

where $\widetilde{\mathbf{y}}_v^{(l)} = \frac{\mathbf{y}_v^{(l)}}{\|\mathbf{y}_v^{(l)}\|_2}$ and $\mathbf{z} \in \mathbb{R}^{d_y}$ is a learning parameter. The score $s_v^{(l)}$ indicates which neighborhood (with its Deformable GConv) is suitable for node v. For each node, Deformable GCN softly chooses a suitable neighborhood with node positional embeddings, and Deformable GConv performs convolution with a deformed convolution kernel.

Loss functions. To learn more effective deformable graph convolution in a latent space without collapsed kernel points, we impose two regularizers: a *separating* regularizer and a *focusing* regularizer. The *separating* regularization loss maximizes the distance between kernel vectors $\{\widetilde{\boldsymbol{\phi}}_k\}_{k=1}^{k=K}$ so that our kernel differentially yields transformation matrices against diverse relations. It is formulated as

$$\mathcal{L}_{sep.} = -\frac{1}{K} \sum_{k_1 \neq k_2} \left\| \widetilde{\boldsymbol{\phi}}_{k_2} - \widetilde{\boldsymbol{\phi}}_{k_1} \right\|_2^2. \quad (9)$$

In addition, to avoid extreme changes of the deformable kernel g_{deform} and the collapse of kernel points after deformations, we use a *focusing* regularizer that penalizes the ℓ_2-norm of deformation vectors $\Delta_k(\mathbf{e}_v)$ as

$$\mathcal{L}_{focus} = \frac{1}{K \cdot |V|} \sum_{v\in V} \sum_{k=1}^{K} \|\Delta_k(\mathbf{e}_v)\|_2^2. \quad (10)$$

With these two regularizer losses, our Graph Deformable Convolutional Network properly generates the kernel vectors and deformation vectors. Since we mainly conduct experiments on node classification tasks, we use the standard cross-entropy loss function $\mathcal{L}_{cls}$ with the two regularizers. The total loss function is as follows:

$$\mathcal{L} = \mathcal{L}_{cls} + \alpha \cdot \mathcal{L}_{sep.} + \beta \cdot \mathcal{L}_{focus}, \quad (11)$$

where α and β are hyperparameters for the strength of regularizations, $\mathcal{L}_{sep.}$ and $\mathcal{L}_{focus}$.

4 Experiments

In this section, we validate the effectiveness of our framework, Deformable GCN, using heterophily and homophily graph datasets.

4.1 Dataset

For validating our model, we use six heterophilic graph datasets and three homophilic graph datasets, which can be distinguished by the *homophily ratio* (Zhu et al. 2020) $h = \frac{|\{(u,v):(u,v)\in E \wedge y_u = y_v\}|}{|E|}$, where y_v is the label of node v. Statistics of each dataset are in Table 1.

Heterophilic graphs. We utilize six datasets for evaluating performance on *heterophilic graphs*, which have low homophily ratio. Texas, Wisconsin, and Cornell are web page datasets (Pei et al. 2020) collected by CMU. Nodes correspond to web pages and edges correspond to hyperlinks between them. Node features are the bag-of-words representations of the web pages and labels are five categories.

Squirrel and Chameleon are web page datasets in Wikipedia (Rozemberczki, Allen, and Sarkar 2021), where the nodes are web pages, edges are links between them, node features are keywords of the pages, and labels are five categories based on the monthly traffic of the web pages.

Actor dataset is a subgraph of the film-director-actor-writer network (Tang et al. 2009). Each node represents an actor, edges represent co-occurrence on the same Wikipedia page, and node features are keywords on the Wikipedia page. The number of node labels is five.

Homophilic graphs. We utilize three standard citation graphs such as Citeseer, Pubmed, and Cora (Sen et al. 2008) for evaluation on *homophilic graphs*. In these graph datasets, nodes represent documents and edges represent citations. Node features are the bag-of-words representations of papers and node labels are the topics of each paper.

4.2 Baselines and Implementation Details

Baselines. For baseline models, we include widely-used GNN-based methods: GCN (Kipf and Welling 2017), GAT (Veličković et al. 2017), and ChebyNet (Defferrard, Bresson, and Vandergheynst 2016). We also include a 2-layer MLP as a baseline since MLP models show comparable performance under heterophily when existing methods for graph-structured data do not use the graph topology effectively. To compare our models with state-of-the-art models on heterophilic graphs, we include JKNet (Xu et al. 2018), MixHop (Abu-El-Haija et al. 2019), H_2GCN (Zhu et al. 2020), and Geom-GCN (Pei et al. 2020). We use the best models among the three variants of Geom-GCN from the paper (Pei et al. 2020).

Implementation details. We use the Adam optimizer (Kingma and Ba 2014) with ℓ_2-regularization and 500 epochs for training our model and the baselines. For all cases, hyperparameters are optimized in the same search space: learning rate in {0.01, 0.005}, weight decay in {1e-3, 5e-4, 5e-5}, and hidden dimension in {32, 64}. Dropout is applied with 0.5 dropout rate. The performance

Dataset	Heterophilic Graphs						Homophilic Graphs		
	Texas	Wisconsin	Actor	Squirrel	Chameleon	Cornell	Citeseer	Pubmed	Cora
# Classes	5	5	5	5	5	5	6	3	7
# Nodes	183	251	7600	5201	2277	183	3327	19717	2708
# Edges	279	450	26659	198353	31371	277	4552	44324	5278
# Features	1703	1703	932	2089	2325	1703	3703	500	1433
Avg deg.	3.05	3.59	7.02	76.28	27.55	3.03	3.03	4.50	3.90
Hom. ratio h	0.11	0.21	0.22	0.22	0.23	0.30	0.74	0.80	0.81

Table 1: Dataset statistics.

Dataset Hom. ratio h	Texas 0.11	Wisconsin 0.21	Actor 0.22	Squirrel 0.22	Chameleon 0.23	Cornell 0.30	Citeseer 0.74	Pubmed 0.80	Cora 0.81
MLP	82.16±2.64	84.90±1.82	36.78±0.56	30.77±1.68	47.87±1.31	81.08±3.62	72.86±1.42	87.62±0.19	75.09±1.45
GCN	64.32±2.60	62.94±4.19	30.47±0.64	46.65±0.99	64.98±0.69	60.27±2.37	76.66±1.12	87.59±0.36	87.44±0.83
GAT	60.81±4.62	64.31±3.48	29.92±0.43	45.47±1.38	66.56±0.99	59.46±1.58	76.54±1.00	86.55±0.34	87.46±0.77
ChebyNet	76.49±3.45	77.84±3.29	35.03±0.73	45.42±0.06	61.80±1.19	72.97±4.61	76.20±1.12	89.16±0.30	83.44±2.04
JKNet	62.97±5.18	60.78±3.29	30.78±0.60	53.78±1.25	68.53±1.59	59.19±2.16	76.05±1.00	88.64±0.34	87.12±0.81
MixHop	83.78±3.35	85.10±2.57	33.80±0.89	39.32±2.16	63.09±1.07	81.08±4.61	75.86±1.20	89.02±0.26	87.20±0.85
Geom-GCN	68.11±3.04	64.51±2.53	31.48±0.64	38.00±0.60	60.99±1.74	60.54±3.55	**77.48±0.87**	**89.51±0.33**	85.51±0.92
H_2GCN	82.16±4.12	86.67±2.18	36.96±0.55	54.51±0.94	65.42±1.58	80.54±3.77	77.05±0.9	89.38±0.26	**87.48±0.93**
Deformable GCN	**84.32±3.42**	**87.06±2.16**	**37.07±0.79**	**62.56±1.31**	**70.90±1.12**	**85.95±2.71**	76.83±1.15	89.49±0.29	**87.48±0.82**

Table 2: Evaluation results on node classification task (Mean accuracy (%) $\pm$ 95% confidence interval). The best-performing models are highlighted with boldface.

is reported with the best model on the validation datasets. For all datasets, we apply the splits (48%/ 32%/ 20%)[1] of nodes per class for (train/ validation/ test) provided by (Pei et al. 2020) for a fair comparison as (Zhu et al. 2020). All experiments are repeated 10 times as (Pei et al. 2020; Zhu et al. 2020) and accuracy is used as an evaluation metric. More implementation details are in the supplementary materials.

4.3 Results on Node Classification

Table 2 shows the results of Deformable GCN and other baselines on node classification tasks. The best model for each dataset is highlighted with boldface. Overall, Deformable GCN achieves the highest performance on all heterophilic graphs compared to all baselines including H_2GCN, which is specifically proposed for heterophilic graphs. Note that on some heterophilic graph datasets such as Texas, Wisconsin, Actor, and Cornell, MLP outperforms various GNNs such as GCN, GAT, and Geom-GCN by significant margins without utilizing any graph structure information. It might seem that graph structure information is harmful to representation learning on heterophilic graphs for most existing GNN models, but on other heterophilic graph datasets such as Squirrel and Chameleon, MLP underperforms most GNN models. In contrast to the existing GNN models, our Deformable Graph Convolution in various *position spaces* with *node positional embeddings* allows Deformable GCN to consistently achieve the best performance on all the heterophilic graph datasets.

Similar to our method, Geom-GCN also performs convolution in a latent space. But it significantly underperforms our method by 17.4% on average on heterophilic graph datasets and especially on Cornell the gap is 25.4%. We believe that our node positional embeddings, which are simultaneously learned with node representations for the target task in an end-to-end fashion, are more effective than the node embeddings in Geom-GCN that are obtained from pretrained external embedding methods. On homophilic graphs, Deformable GCN shows comparable performance. Since most existing GNNs have been proposed based on the homophily assumption, the performance gap between GNN-based models is small.

4.4 Ablation Study and Analysis

We conduct additional experiments to verify the contributions of our node positional embedding, deformable graph convolution layer, deformation, and regularization for Deformable GCN and analyze the attention score $\left\{s^{(l)}\right\}_l$ and the receptive field of Deformable GCN.

Node positional embedding. To verify the efficacy of our node positional embedding method, we compare it with other node embedding methods such as node2vec (Grover and Leskovec 2016) and Poincare embedding (Nickel and Kiela 2017) by using them for ϕ_v in (4) on four datasets in Table 3. The result shows that our proposed embedding

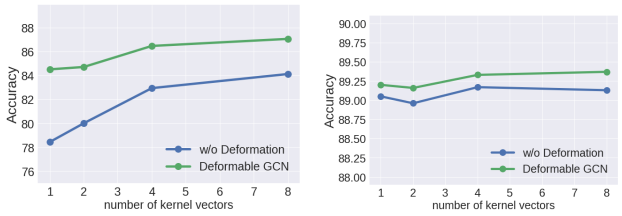
[1]https://github.com/graphdml-uiuc-jlu/geom-gcn.

Positional embedding	Dataset			
	Wisconsin	Actor	Squirrel	Pubmed
node2vec	67.57	35.04	45.27	88.35
PoinCare	68.1	35.15	46.31	87.26
Ours	**87.06**	**37.07**	**62.56**	**89.49**

Table 3: Comparison of our node positional embeddings with other node embedding methods on four datasets.

Layer	Dataset			
	Wisconsin	Actor	Squirrel	Pubmed
GAT Layer	70.54	36.26	61.62	88.04
Deformable GConv	**87.06**	**37.07**	**62.56**	**89.49**

Table 4: Comparison of our Deformable GConv with GAT Layer on four datasets.



(a) Wisconsin (b) Pubmed

Figure 2: Ablations for deformation of deformable graph convolution. We compare **Deformable GCN** and the model without deformation (**w/o Deformation**), which uses only kernel vectors on (a) Wisconsin and (b) Pubmed datasets according to number of the kernel vectors.

method outperforms two other embedding methods for each dataset. Specifically, our proposed embedding method improves over node2vec and Poincare embedding by 9.9% and 9.8% on average, respectively. Unlike node2vec and Poincare embeddings that require separate pre-training, our positional embedding can be simultaneously trained with GNNs in an end-to-end fashion. We believe that our embedding scheme is much easier to train and allows more optimized positional embeddings for target tasks.

Deformable graph convolution. In Table 4, we conduct an experiment to examine the contribution of deformable graph convolution (Deformable GConv) by substituting it with the GAT Layer in Graph Attention Networks (Veličković et al. 2017) on four datasets. From the table, Deformable GConv consistently shows better performance compared to GAT Layer. In particular, on Wisconsin dataset, Deformable GConv improves 16.52% over GAT Layer. It means that Deformable GConv contributes to the performance improvements of Deformable GCN.

Deformation. To reveal the effectiveness of deformation, we compare the results of Deformable GCN and the model without deformation (w/o Deformation) on node classification task with Wisconsin and Pubmed datasets accord-

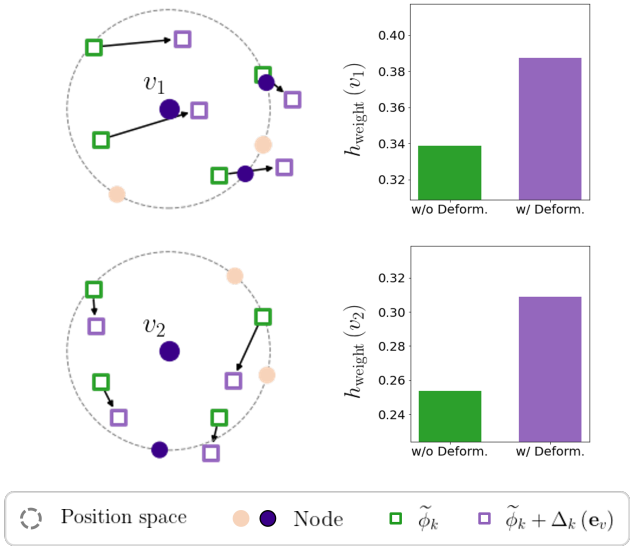


Figure 3: (Left) The visualization of kernel vectors without deformation $\widetilde{\phi}_k$ and kernel vectors with deformation $\widetilde{\phi}_k + \Delta_k(\mathbf{e}_v)$ in a position space centered at each node v_1 and v_2. The color of a node indicates its label. The kernel vectors are differentially deformed to increase the homophilic weight $h_{\text{weight}}(v)$ approaching neighbor nodes with the same labels (black solid circles). (Right) Comparisons of the homophilic weight $h_{\text{weight}}(v)$ defined in (12) between w/o and w/ Deformation.

ing to the number of kernel vectors. Figure 2 shows that Deformable GCNs consistently achieve superior performance ($2 \sim 6\%$ on average) than the models without deformation (w/o Deformation) on both a heterophilic graph Wisconsin and a homophilic graph Pubmed in the various settings with different numbers of kernel points.

Also, for qualitatively analyzing the effectiveness of deformation, we visualize the kernel vectors and deformation in the position space in Figure 3. To study the efficacy of the deformation, we calculate the homophilic weight based on $a_{u,v,k}$ in (6) as follows:

$$h_{\text{weight}}(v) = \sum_{\{u:u\in N(v)\wedge y_u=y_v\}} \sum_{k} a_{u,v,k}. \qquad (12)$$

It indicates the summation of the weight $a_{u,v,k}$ of neighbor nodes with the same label as center node v. In Figure 3, we compare $h_{\text{weight}}(v)$ before/after deformation. After deformation, the kernel vectors tend to become closer to the neighbor nodes with the same labels, which are black solid circles. As a result $h_{\text{weight}}(v)$ becomes larger values and Deformable GCN receives more messages from meaningful neighborhoods (e.g., with the same labels).

Regularizers. To verify contribution of regularizers, we conduct an ablation study on regularizers such as $\mathcal{L}_{sep.}$ and $\mathcal{L}_{focus}$ on four datasets. Table 5 summarizes the re-

Regularizer		Dataset			
$\mathcal{L}_{sep.}$	$\mathcal{L}_{focus}$	Wisconsin	Actor	Squirrel	Pubmed
		84.12	36.67	60.31	89.02
✓		86.08	36.70	60.43	88.86
	✓	86.08	36.67	61.83	88.79
✓	✓	**87.06**	**37.07**	**62.56**	**89.49**

Table 5: Ablations for two regularizer losses ($\mathcal{L}_{sep.}, \mathcal{L}_{focus}$) on four datasets.

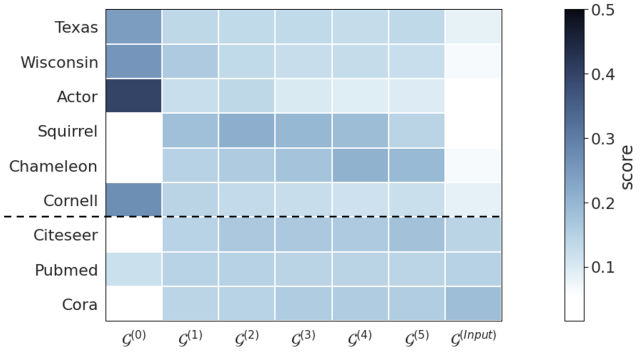


Figure 4: The attention score $s^{(l)}$ of each *latent neighborhood graph* $\mathcal{G}^{(l)}$. Datasets above the dash line are heterophilic graphs and the others homophilic graph datasets.

sults of the ablation study of our regularizers. From the table, each regularizer contributes to improving performance of Deformable GCN. Moreover, we observe that training with both regularizers is the most effective. On average, Deformable GCNs trained with both regularizers outperform ones without regularizers by 1.5%.

Attention score $s_v^{(l)}$. The attention score $s_v^{(l)}$ in (8) at node v for a latent neighborhood graph $\mathcal{G}^{(l)}$ can be used to understand datasets. An averaged attention score $s^{(l)} = \frac{1}{|V|}\sum_{v\in V} s_v^{(l)}$ indicates the overall importance of a latent neighborhood graph $\mathcal{G}^{(l)}$. Figure 4 shows the attention score of a Deformable GCN with $L = 5$ that utilizes 6 latent neighborhood graphs $\{\mathcal{G}^{(l)}\}_{l=0}^{l=5}$ and a original input graph $\mathcal{G}^{(\text{Input})}$. On most heterophilic graph datasets except for Chameleon and Squirrel, Deformable GCN has the large attention score for latent graph $\mathcal{G}^{(0)}$ and relatively small attention scores for $\mathcal{G}^{(\text{Input})}$. Recall that $\mathcal{G}^{(0)}$ is the kNN graph constructed based on the similarity between input features. This indicates that rather than focusing on the neighborhood on the input graph, smoothing over the nodes with similar input features is more helpful for predictions on heterophilic graphs. Indeed, this coincides with the results in Table 2. On the heterophilic graphs such as Texas, Wisconsin, Actor, and Cornell, an MLP, which does *not* use any graph structure information, outperforms most existing GNNs. On the other hand on homophilic graphs, $\mathcal{G}^{(\text{Input})}$ plays a more important role on homophilic graphs compared to heterophilic graphs.

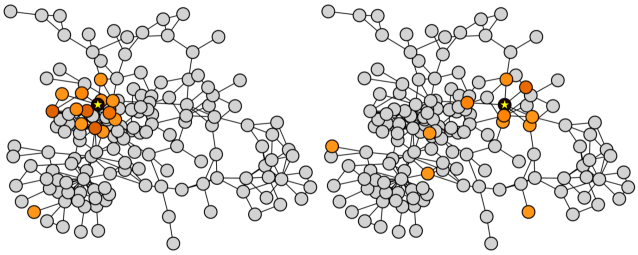


Figure 5: The visualization of the receptive field of each target node marked with star on Cora dataset. The intensity of the color of each node is associated with the intensity of each node.

Receptive field of Deformable GCN. We visualize the receptive field for each target node, which is marked as yellow star, on the subgraph of Cora dataset in Figure 5. For visualization, we compute the intensity of node u as $\sum_l \sum_k s_v^{(l)} \cdot \widehat{a}_{u,v,k}$, where the node v is target node. We represent the intensity of the color of each node according to the intensity of each node. The figure shows that the receptive field of Deformable GCN varies depending on the target node on a single graph, which means that our model flexibly captures various ranges of node depenedencies as needed.

5 Conclusion

We proposed Deformable Graph Convolutional Networks for learning representations on both heterophilic and homophilic graphs. Our approach learns node positional embeddings for mapping nodes into latent spaces and applies deformable graph convolution on each node. The convolution kernels in deformable graph convolution are transformed by shifting kernel vectors. Our experiments show that the Deformable Graph Convolution Networks are effective for learning representations on both heterophilic and homophilic graphs. Future directions include studying the effectiveness of Deformable GCN in other tasks such as link prediction and graph classification and expanding Deformable GConv to other domains.

Limitations. In our work, for all datasets, we generate the node positional embeddings with a simple procedure that is smoothing input features followed by a linear transformation. We believe that for the datasets with more complicated input features, the simple procedure may not be sufficient and more advanced techniques might need to be developed.

Acknowledgments

This work was partly supported by MSIT (Ministry of Science and ICT), Korea, under the ICT Creative Consilience program (IITP-2021-2020-0-01819) supervised by the IITP and Samsung Research Funding & Incubation Center of Samsung Electronics under Project Number SRFC-IT1701-51.

References

Abu-El-Haija, S.; Perozzi, B.; Kapoor, A.; Alipourfard, N.; Lerman, K.; Harutyunyan, H.; Ver Steeg, G.; and Galstyan, A. 2019. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *ICML*.

Berg, R. v. d.; Kipf, T. N.; and Welling, M. 2017. Graph convolutional matrix completion. arXiv:1706.02263.

Bo, D.; Wang, X.; Shi, C.; and Shen, H. 2021. Beyond Low-frequency Information in Graph Convolutional Networks. In *AAAI*.

Dai, J.; Qi, H.; Xiong, Y.; Li, Y.; Zhang, G.; Hu, H.; and Wei, Y. 2017. Deformable Convolutional Networks. In *ICCV*.

Defferrard, M.; Bresson, X.; and Vandergheynst, P. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *NeurIPS*.

Erkan, G.; and Radev, D. R. 2004. Lexrank: Graph-based lexical centrality as salience in text summarization. *JAIR*, 22: 457–479.

Errica, F.; Podda, M.; Bacciu, D.; and Micheli, A. 2019. A fair comparison of graph neural networks for graph classification. In *ICLR*.

Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; and Dahl, G. E. 2017. Neural message passing for quantum chemistry. In *ICML*.

Grover, A.; and Leskovec, J. 2016. node2vec: Scalable feature learning for networks. In *KDD*.

Hamilton, W. L.; Ying, R.; and Leskovec, J. 2018. Inductive Representation Learning on Large Graphs. In *NeurIPS*.

Hammond, D. K.; Vandergheynst, P.; and Gribonval, R. 2011. Wavelets on graphs via spectral graph theory. *ACHA*, 30(2): 129–150.

He, K.; Gkioxari, G.; Dollár, P.; and Girshick, R. 2017. Mask r-cnn. In *ICCV*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *CVPR*.

Johnson, J.; Krishna, R.; Stark, M.; Li, L.-J.; Shamma, D.; Bernstein, M.; and Fei-Fei, L. 2015. Image retrieval using scene graphs. In *CVPR*.

Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. In *ICLR*.

Kipf, T. N.; and Welling, M. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.

Li, P.; Wang, Y.; Wang, H.; and Leskovec, J. 2020. Distance Encoding–Design Provably More Powerful GNNs for Structural Representation Learning. In *NeurIPS*.

Li, Q.; Han, Z.; and Wu, X.-M. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI*.

Liu, M.; Wang, Z.; and Ji, S. 2020. Non-local graph neural networks. arXiv:2005.14612.

McPherson, M.; Smith-Lovin, L.; and Cook, J. M. 2001. Birds of a feather: Homophily in social networks. *Annu. Rev. Sociol.*, 27(1): 415–444.

Nickel, M.; and Kiela, D. 2017. Poincaré embeddings for learning hierarchical representations. In *NeurIPS*.

Pei, H.; Wei, B.; Chang, K. C.-C.; Lei, Y.; and Yang, B. 2020. Geom-gcn: Geometric graph convolutional networks. In *ICLR*.

Rozemberczki, B.; Allen, C.; and Sarkar, R. 2021. Multi-scale attributed node embedding. *J. Complex Netw.*, 9(2).

Schlichtkrull, M.; Kipf, T. N.; Bloem, P.; Van Den Berg, R.; Titov, I.; and Welling, M. 2018. Modeling relational data with graph convolutional networks. In *ESWC*, 593–607.

Sen, P.; Namata, G.; Bilgic, M.; Getoor, L.; Galligher, B.; and Eliassi-Rad, T. 2008. Collective classification in network data. *AI magazine*, 29(3): 93–93.

Siarohin, A.; Sangineto, E.; Lathuilière, S.; and Sebe, N. 2018. Deformable GANs for Pose-Based Human Image Generation. In *CVPR*.

Tang, J.; Qu, M.; Wang, M.; Zhang, M.; Yan, J.; and Mei, Q. 2015. Line: Large-scale information network embedding. In *WWW*.

Tang, J.; Sun, J.; Wang, C.; and Yang, Z. 2009. Social influence analysis in large-scale networks. In *SIGKDD*.

Thomas, H.; Qi, C. R.; Deschaud, J.-E.; Marcotegui, B.; Goulette, F.; and Guibas, L. J. 2019. Kpconv: Flexible and deformable convolution for point clouds. In *ICCV*.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention is all you need. In *NeurIPS*.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; and Bengio, Y. 2017. Graph attention networks. In *ICLR*.

Wang, D.; Cui, P.; and Zhu, W. 2016. Structural deep network embedding. In *KDD*.

Wang, X.; Chan, K. C. K.; Yu, K.; Dong, C.; and Loy, C. C. 2019. EDVR: Video Restoration With Enhanced Deformable Convolutional Networks. In *CVPR W*.

Wu, F.; Souza, A.; Zhang, T.; Fifty, C.; Yu, T.; and Weinberger, K. 2019. Simplifying graph convolutional networks. In *ICML*.

Xu, K.; Li, C.; Tian, Y.; Sonobe, T.; Kawarabayashi, K.-i.; and Jegelka, S. 2018. Representation learning on graphs with jumping knowledge networks. In *ICML*.

Ying, R.; You, J.; Morris, C.; Ren, X.; Hamilton, W. L.; and Leskovec, J. 2018. Hierarchical graph representation learning with differentiable pooling. In *NeurIPS*.

Yun, S.; Jeong, M.; Kim, R.; Kang, J.; and Kim, H. J. 2019. Graph Transformer Networks. In *NeurIPS*.

Zhang, M.; and Chen, Y. 2018. Link prediction based on graph neural networks. In *NeurIPS*.

Zhu, J.; Rossi, R. A.; Rao, A.; Mai, T.; Lipka, N.; Ahmed, N. K.; and Koutra, D. 2021. Graph Neural Networks with Heterophily. In *AAAI*.

Zhu, J.; Yan, Y.; Zhao, L.; Heimann, M.; Akoglu, L.; and Koutra, D. 2020. Beyond Homophily in Graph Neural Networks: Current Limitations and Effective Designs. In *NeurIPS*.

Zhu, X.; Hu, H.; Lin, S.; and Dai, J. 2019. Deformable convnets v2: More deformable, better results. In *CVPR*.