11
Graph Neural Networks in Computer Vision

11.1 Introduction

Graph-structured data widely exists in numerous tasks in the area of computer vision. In the task of visual question answering, where a question is required to be answered based on content in a given image, graphs can be utilized to model the relations among the objects in the image. In the task of skeleton-based recognition, where the goal is to predict human action based on the skeleton dynamics, the skeletons can be represented as graphs. In image classification, different categories are related to each other through knowledge graphs or category co-occurrence graphs (Wang et al., 2018b; Chen et al., 2019c). Furthermore, point cloud, which is a type of irregular data structure representing shapes and objects, can also be denoted as graphs. Therefore, graph neural networks can be naturally utilized to extract patterns from these graphs to facilitate the corresponding computer vision tasks. This chapter demonstrates how graph neural networks can be adapted to the aforementioned computer vision tasks with representative algorithms.

11.2 Visual Question Answering

Given an image and a question described in natural language, the task of visual question answering (VQA) is to answer the question based on the information provided in the image. An illustrative example of the VQA task is shown in Figure 11.1, where the task is to figure out the color of the fruit at the left of the image. To perform the VQA task properly, it is necessary to understand the question and the image, which requires techniques from both natural language processing and computer vision. Typically, Convolutional Neural Networks (CNNs) are adopted to learn the image representation. Then, it is combined

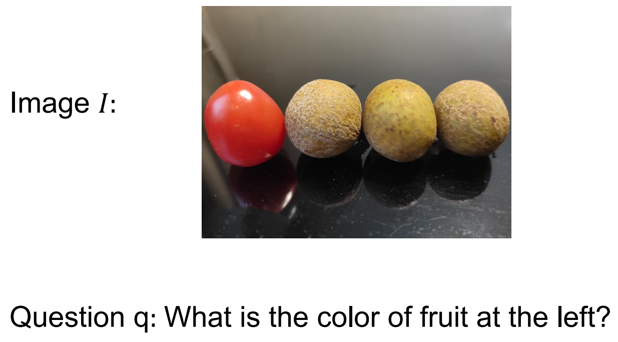

Figure 11.1 An illustrative example of the VQA task

with the representation of the question to perform the VQA task. As illustrated in Figure 11.1, the relations between objects in the image can also be important to answer the question correctly. Better capturing the semantic and spatial relations between the objects can potentially facilitate the VQA task. For example, to answer the question q in Figure 11.1 properly, the relative locations between the fruits are necessary. To denote the objects' interactions explicitly, graphs are adopted to model the connections between objects. Graph neural networks are then adopted to learn the representations for these graphs generated from images (Teney et al., 2017; Norcliffe-Brown et al., 2018). Specifically, some works assume that the graph is given for each image (Teney et al., 2017), while others incorporate the graph generation process as a part of the proposed models (Norcliffe-Brown et al., 2018). In this section, we introduce the two models proposed in (Teney et al., 2017) and (Norcliffe-Brown et al., 2018) as examples to show how graph neural networks can be adopted in the VQA task.

The task of VQA is modeled as a classification problem, where each class corresponds to one of the most common answers in the training set. Formally, each sample of this classification problem can be denoted as (q, I), where q is the question, and I is the image. To tackle this classification problem utilizing the information from the question and the image, their representations are learned and combined to serve as the input for the prediction layer based on feedforward network. In (Norcliffe-Brown et al., 2018), the image is transferred to a graph in an end-to-end manner while training the entire framework. In (Teney et al., 2017), both the question q and the image I are pre-processed as graphs and dealt with graph neural networks.

11.2.1 Images as Graphs

In (Norcliffe-Brown et al., 2018), the question q is encoded to a representation $\mathbf{q}$ using RNN with GRU. To learn the representation for the image I, a graph is generated from the image I dependent on the question q, and a graph neural network model is applied on the generated graph to learn its representation. Next, we first describe how the graph is generated given the image I and the question representation $\mathbf{q}$. Then, we introduce the graph neural network model to learn the graph representation $\mathbf{F}_I$. Finally, we briefly describe the prediction layer, which takes the representations for the question q and the image I as input.

Given an image I, and a set of n visual features bounded by boxes generated by an object detector. Each bounding box serves as a node in the generated graph. An initial representation $\mathbf{x}_i$ is produced for each node v_i by taking the average of the corresponding convolutional feature maps in the bounding box. These nodes consist of the node set for the generated graph, denoted as $\mathcal{V}_I$. We then generate the set of edges $\mathcal{E}_I$ to describe the relations between these nodes. These edges are constructed based on the pair-wise similarity and the relevance to the given question q. To combine these two types of information, for each node, a question-dependent representation is generated as:

$$\mathbf{e}_i = h([\mathbf{x}_i, \mathbf{q}]),$$

where $\mathbf{e}_i$ is the question-dependent node representation for node v_i and $h()$ is a non-linear function to combine these two types of information. The question-dependent representations for all nodes can be summarized by a matrix $\mathbf{E}$ where the i-th row is corresponding to the i-th node in the generated graph. Then the adjacency matrix of the graph is calculated as:

$$\mathbf{A} = \mathbf{E}\mathbf{E}^T. \tag{11.1}$$

However, the adjacency matrix learned by Eq. (11.1) is fully connected, which is not optimal for both efficiency and the performance of the model. Hence, to generate a sparse adjacency matrix, only the stronger connections for each node are kept. Specifically, we only keep the top m values of each row and set other values to 0, where m is a hyper-parameter. The graph generated for image I is denoted as $\mathcal{G}_I$.

After obtaining the question-dependent graph $\mathcal{G}_I$ for the objects detected in the image, the Mo-Filter introduced in Section 5.3.2 is adapted to generate the node representations. The operation for a node v_i can be formulated as:

$$\mathbf{F}_i^{(l)} = \sum_{v_j \in \mathcal{N}(v_i)} w(\mathbf{u}(i, j))\mathbf{F}_j^{(l-1)}\alpha_{i,j}, \tag{11.2}$$

where $\mathcal{N}(v_i)$ denotes the set of neighbors for node v_i, $w(\cdot)$ is a learnable Gaussian kernel, $\alpha_{i,j} = \text{softmax}(\mathbf{A}[\text{i,:}])[j]$ indicates the strength of connectivity between nodes v_i and v_j, and $\mathbf{u}(i, j)$ is a pseudo-coordinate function. This pseudo-coordinate function $\mathbf{u}(i, j)$ returns a polar coordinate vector (ρ, θ), which describes the relative spatial positions of the centers of the bounding boxes corresponding to nodes v_i and v_j. After applying L consecutive graph filtering layers as described in Eq. (11.2), the final representation for each node v_i is obtained as $\mathbf{F}_i^{(L)}$. In (Norcliffe-Brown et al., 2018), K different Gaussian kernels are used and the output representations of the K kernels are combined as:

$$\mathbf{F}_i = \|_{k=1}^{K} \mathbf{F}_{i,k}^{(L)} \mathbf{\Theta}_k,$$

where $\mathbf{\Theta}_k$ is a learnable linear transformation. The final representations for all the nodes in the graph can be summarized in a matrix $\mathbf{F}$ where each row corresponds to a node.

Once these final node representations are obtained, a max-pooling layer is applied to generate the representation $\mathbf{F}_I$ for the graph $\mathcal{G}_I$. The graph representation $\mathbf{F}_I$ and the question representation $\mathbf{q}$ are combined through the element-wise product to generate the task representation, which is then input into the feedforward network-based prediction layer to perform the classification.

11.2.2 Images and Questions as Graphs

In (Teney et al., 2017), both the question q and the image I are pre-processed as graphs. The question q is modeled as a syntactic dependency tree. In the tree, each word in the sentence is a node, and the dependency relations between words are edges. We denote the graph generated for a question q as $\mathcal{G}_q = \{\mathcal{V}_q, \mathcal{E}_q, \mathcal{R}_q\}$, where $\mathcal{R}_q$ is the set of possible dependency relations. Meanwhile, the image I is pre-processed as a fully connected graph. In the graph, the objects in the image I are extracted as nodes, and they are pair-wisely connected. We denote the graph generated for the image I as $\mathcal{G}_I = \{\mathcal{V}_I, \mathcal{E}_I\}$. Each object (or node) $v_i \in \mathcal{V}_I$, is associated with its visual features $\mathbf{x}_i$ while each edge $(v_i, v_j) \in \mathcal{E}_I$ between nodes v_i and v_j is associated with a vector $\mathbf{x}_{ij}$ that encodes the relative spatial relations between v_i and v_j.

Both graphs are processed with graph neural networks to generate node representations, which are later combined to generate a representation for the pair (q, I). In (Teney et al., 2017), a slightly modified version of GGNN-Filter as introduced in Section 5.3.2, is utilized to process these two graphs. The modified

GGNN-Filter can be described as:

$$\mathbf{m}_i = \sum_{v_j \in \mathcal{N}(v_i)} \mathbf{x}'_{ij} \odot \mathbf{x}'_j, \tag{11.3}$$

$$\mathbf{h}_i^{(t)} = \text{GRU}([\mathbf{m}_i, \mathbf{x}'_i], \mathbf{h}_i^{(t-1)}); t = 1, \dots T, \tag{11.4}$$

where $\mathbf{x}'_j$ and $\mathbf{x}'_{ij}$ are the features for node v_j and edge (v_i, v_j), respectively. For the question graph $\mathcal{G}_q$, $\mathbf{x}'_j$ and $\mathbf{x}'_{ij}$ are randomly initialized. In detail, node features are word-specific, i.e., each word is initialized with a representation while edge features are relation specific, i.e., edges with the same relation $r \in \mathcal{R}_q$ share the same features. For the image graph $\mathcal{G}_I$, $\mathbf{x}'_i$ and $\mathbf{x}'_{ij}$ are transformed using feedforward networks from the associated features $\mathbf{x}_i$ and $\mathbf{x}_{ij}$, respectively. In Eq. (11.4), the GRU update unit (with $\mathbf{h}_0^0 = 0$) runs T times and finally obtains the final representation $\mathbf{h}_i^{(T)}$ for node v_i. Note that, in (Teney et al., 2017), a single layer of graph filter as described in Eq. (11.3) and Eq. (11.4) is utilized to process the graphs. In other words, there are a single aggregation step and T GRU update steps. We denote the final node representations learned from the graph filtering as $\mathbf{h}_i^{(T,q)}$ and $\mathbf{h}_j^{(T,I)}$ for node $v_i \in \mathcal{V}_q$ in the question graph $\mathcal{G}_q$ and $v_j \in \mathcal{V}_I$ in the image graph $\mathcal{G}_I$, respectively. These node representations from the two graphs are combined as:

$$\mathbf{h}_{i,j} = \alpha_{i,j} \cdot [\mathbf{h}_i^{(T,q)}, \mathbf{h}_j^{(T,I)}], i = 1, \dots |\mathcal{V}_q|; j = 1, \dots, |\mathcal{V}_I|, \tag{11.5}$$

$$\mathbf{h}'_i = f_1\left(\sum_{j=1}^{|\mathcal{V}_I|} \mathbf{h}_{i,j}\right), \tag{11.6}$$

$$\mathbf{h}_{(q,I)} = f_2\left(\sum_{i=1}^{|\mathcal{V}_q|} \mathbf{h}'_i\right), \tag{11.7}$$

where $\alpha_{i,j}$ in Eq. (11.5), which is learned using the raw features $\mathbf{x}'$, can be regarded as a relevance measure between a question node and an image node. Specifically, it can be modeled as:

$$\alpha_{i,j} = \sigma\left(f_3\left(\frac{\mathbf{x}_i'^{Q}}{\|\mathbf{x}_i'^{Q}\|} \odot \frac{\mathbf{x}_j'^{I}}{\|\mathbf{x}_j'^{I}\|}\right)\right),$$

where we use the superscripts Q and I to differentiate the features for nodes from the question graph and the image graph respectively, $\odot$ is the Hadarmard product, $f_3()$ is modeled as a linear transformation and $\sigma()$ is the sigmoid function. $\mathbf{h}_{i,j}$ is a mixed representation of a node from the question graph and a node from the image graph. These representations $\mathbf{h}_{i,j}$ are hierarchically aggregated to generate the representation $\mathbf{h}_{(q,I)}$ for the pair (q, I) in Eq. (11.6) and

Eq. (11.7), where $f_1()$ and $f_2()$ are feedforward neural networks. The representation can be utilized to perform the classification on the candidate sets.

11.3 Skeleton-based Action Recognition

Human action recognition is an active research area, which plays a vital role in video understanding. Human body skeleton dynamics can capture important information about human actions, which have been often leveraged for action recognition. The skeleton dynamics can be naturally modeled as a time series of human joint locations and interactions between them. Especially, the spatial relations between the joints can be modeled as a graph with the joints as the nodes and bones as edges connecting them. Then, the skeleton dynamics can be represented as a sequence of graphs which share the same spatial structure while the node attributes (or location coordinates of the joints) of the graph in the sequence are different. Graph neural networks have been adopted to learn better representations of the skeleton dynamics and thus improve the performance of skeleton-based action recognition (Yan et al., 2018; Li et al., 2018a; Shi et al., 2019a; Si et al., 2018; Wen et al., 2019; Li et al., 2019c; Si et al., 2019). In this section, we take the framework proposed in (Yan et al., 2018) as one example to demonstrate how graph neural networks can be applied to the skeleton-based action recognition task. It is the first to explore graph neural networks for skeleton-based action recognition.

As shown in Figure 11.2, a sequence of skeletons is represented as a spatial-temporal graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V}$ denotes the set of nodes and $\mathcal{E}$ is the set of edges, respectively. The node set $\mathcal{V}$ consists of all the joints in the skeleton sequence, i.e., $\mathcal{V} = \{v_{ti} | t = 1, \ldots, T; i = 1, \ldots, N.\}$ where N is the number of joints in a single skeleton graph, and T is the number of skeletons in the sequence. The edge set $\mathcal{E}$ consists of two types of edges: 1) the intra-skeleton edges within the same skeleton, which are defined based on the bones between the joints; and 2) the inter-frame edges, which connect the same joints in consecutive skeletons in the sequence. For the illustrative example in Figure 11.2, the intra-skeleton edges are highlighted by green while the inter-skeleton edges are shown in blue. The skeleton-based action recognition task can then be converted into a graph classification task where the classes are the actions to predict, such as running. To perform this graph classification task, a graph filtering operation is proposed for the spatial-temporal graph to learn node representations. After the node representations are learned, the graph representation is obtained by applying a global pooling layer, such as max-pooling. The graph representation is then utilized as the input to the feedforward networks-based

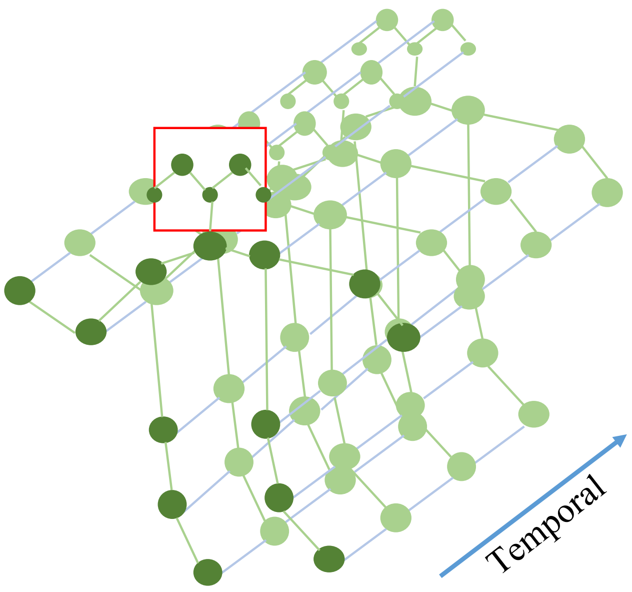

Figure 11.2 An illustrative example of the spatial-temporal skeleton graph

prediction layer. Next, we present the details of the proposed graph filter for the spatial-temporal graph.

The proposed graph filter is adapted from the GCN-Filter (see Section 5.3.2 for details of GCN-Filter), which aggregates information from neighboring nodes in the spatial-temporal graph. Specifically, for a node v_{ti} in the t-th skeleton, its spatial neighbors $\mathcal{N}(v_{ti})$ consist of its 1-hop neighbors in the t-th skeleton graph and the node v_{ti} itself. Then its spatial temporal neighbors $\mathcal{N}^T(v_{ti})$ on the spatial-temporal graph $\mathcal{G}$ can be defined as:

$$\mathcal{N}^T(v_{ti}) = \{v_{\tau j} | \, v_{tj} \in \mathcal{N}(v_{ti}) \text{ and } |\tau - t| \leq \Gamma\}. \tag{11.8}$$

The constraint $|\tau - t| \leq \Gamma$ in Eq. (11.8) indicates that the temporal distance between these two skeleton graphs where the nodes $v_{\tau j}$ and v_{tj} locate should be smaller than Γ. Hence, the spatial temporal neighbors $\mathcal{N}^T(v_{ti})$ of node v_{ti} include not only its spatial neighbors from the same skeleton but also "temporal neighbors" from close skeletons in the sequence. Furthermore, instead of treating the neighbors equally, neighbors are split into different subsets and different transformation matrices are utilized for their transformation. In particular, the spatial neighbors $\mathcal{N}(v_{ti})$ of a node v_{ti} in a skeleton graph are divided to three subsets as follows: 1) the root node itself (i.e., node v_{ti}); 2) the neighboring nodes that are closer to the gravity center of skeleton than the root node; and 3) all other nodes. The neighboring nodes of node v_{ti} in other skeleton graphs can be divided similarly; hence, the neighboring set $\mathcal{N}^T(v_{ti})$ can be divided into $3\Gamma + 1$ sets. For convenience, we use $s(v_{\tau j})$ to indicate the subset a given node $v_{\tau j} \in \mathcal{N}^T(v_{ti})$ belongs to. Then, the graph filtering process for a given node v_{ti} can be described as:

$$\mathbf{F}_{ti}^{(l)} = \sum_{v_{\tau j} \in \mathcal{N}^T(v_{ti})} \frac{1}{\#s(v_{\tau j})} \cdot \mathbf{F}_{\tau j}^{(l-1)} \mathbf{\Theta}_{s(v_{\tau j})}^{(l-1)}, \tag{11.9}$$

where $\mathbf{F}_{ti}^{(l)}$ and $\mathbf{F}_{ti}^{(l-1)}$ denotes the output and input node representations, respectively. $\#s(v_{\tau j})$ denotes the number of neighbors that are in the subset $v_{\tau j}$ and the transformation parameter $\mathbf{\Theta}_{s(v_{\tau j})}^{(l-1)}$ is shared by all the neighbors belonging to the subset $s(v_{\tau j})$. The node representations are learned by stacking L graph filtering layers as Eq. (11.9) with activation layers. Then, the graph representation is obtained by applying a global pooling layer to these node representations. Note that in the framework we introduced above, the relations between the joints in the skeleton are naturally defined through the bones. Hence, only spatially close joints are connected to each other. However, it is possible that some distant joints are also related especially when doing some specific actions. For example, two hands are highly related to each other when doing the action "clapping hands". Thus, it is important to also encode relations between distant joints. In (Shi et al., 2019b,a; Li et al., 2019c), the graphs between the joints are learned together with the parameters of the model.

11.4 Image Classification

Image classification aims to classify an image into certain categories. Graph neural networks have been adopted to advance image classification, especially under zero-shot, few-shot and multi-label settings. In this section, we discuss GNN based image classification under these three settings with representative algorithms. As shown in Figure 3.11 in Section 3.3.5, a CNN-based image classifier usually consists of two parts: 1) feature extraction, which is built with convolutional and pooling layers; and 2) the classification component, which is typically modeled as a fully connected layer. Specifically, this fully connected layer (without considering the softmax layer) can be represented as a matrix $\mathbf{W} \in \mathbb{R}^{d \times c}$, where d is the dimension of the extracted features, and c is the number of categories in the task. The i-th row of $\mathbf{W}$ denoted as $\mathbf{w}_i$ is corresponding to the i-th category, which indicates how likely a given sample is classified to the i-th category. In this section, we loosely call $\mathbf{w}_i$ as the "classifier" of the i-th category.

11.4.1 Zero-shot Image Classification

In the traditional setting of the image classification task in computer vision, abundant images of each category are assumed to be available for training the classifiers for these categories. These learned classifiers can only recognize the images from the categories they are trained with. To recognize images from a new category, thousands of images of this category are required, and their corresponding classifiers must be retrained together with newly collected images. The task of zero-shot image classification is to learn a classifier for a new category without any training images but only based on information about the category, such as its description or its relations with other categories. Graph neural networks are adopted in (Wang et al., 2018b) to learn classifiers for categories without any training images by propagating information from other categories through a knowledge graph describing the relations between categories. Next, we first formally describe the setting of the zero-shot image classification task and then present how graph neural networks are adopted to tackle this problem.

In the zero-shot image classification setting, we are given a set of n categories, among which the first m of them have sufficient training images while the remaining $n - m$ categories are with no images. Each category c_i is associated with a short description, which can be projected to a semantic embedding $\mathbf{x}_i$. Furthermore, there is a knowledge graph (e.g., WordNet) $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ describing the relations between these categories, where the categories are the nodes. In the introduction of this section, we use $\mathbf{w}_i$ to loosely denote a "classifier" of a category c_i. For a linear classifier such as logistic regression for a given category c_i, it can be also represented by its parameters $\mathbf{w}_i \in \mathbb{R}^d$, where d is the dimension of the features of the input image. Given an image, its features can be extracted using some pre-trained Convolutional Neural Networks. For those m categories with sufficient training samples, their corresponding classifier can be learned from these training samples. The goal of the zero-shot image classification task is to learn classifiers for those $n - m$ categories without any images by leveraging their semantic embeddings and/or the given knowledge graph $\mathcal{G}$.

A straightforward way to predict the classifiers is to adopt a neural network that takes the semantic embedding of a category as input and produces its corresponding classifier as output. However, in practice, the number of categories with sufficient training samples is generally too small (e.g., in the order of hundreds) to train the neural network. Hence, instead of deep neural networks, the graph neural network model is adopted to predict the classifiers. The graph neural network model is applied to the knowledge graph with the semantic embeddings of categories as input, and its output is the corresponding classifiers of these categories. In (Wang et al., 2018b), GCN-Filter (see Section 5.3.2 for details on GCN-Filter) is adopted as the graph filtering operation, and L graph filtering layers are stacked to refine the features (with the semantic embeddings as the initial features) before finally obtaining the classifiers. Specifically, the task can be modeled as a regression problem, where the classifiers $\{\mathbf{w}_1, \ldots, \mathbf{w}_m\}$ for the first m categories are served as the ground truth. In (Wang et al., 2018b), the number of layers L is set to a relatively large number (e.g., 6) such that distant information can be propagated through the knowledge graph. However, it is empirically shown that increasing the number of layers of graph neural networks may hurt the performance (Kampffmeyer et al., 2019). Hence, to propagate distant information without reducing the performance, a dense graph is constructed from the given knowledge graph. Any given node is connected to all its ancestors in the knowledge graph. Two graph filtering layers are applied based on the constructed dense graph. In the first layer, information is only aggregated from descendants to ancestors, while in the second layer, information flows from ancestors to descendants.

11.4.2 Few-shot Image Classification

In zero-shot learning, we aim to learn classifiers for unseen categories without any training samples. In the setting of few-shot learning image classification, we are given a set of n categories, among which the first m categories have sufficient training images while the remaining $n - m$ categories are with only k labeled images, where k is usually a very small number such as 3. Specifically, when $k = 0$, it can be treated as the the zero-shot image classification task. In this section, we specifically focus on the case where $k > 0$.

In few-shot image classification, as all categories have labeled images (either sufficient or not), classifiers can be learned for all categories. We denote the classifier learned for the i-th category as $\mathbf{w}_i$. The classifiers $\{\mathbf{w}_1, \ldots, \mathbf{w}_m\}$ learned for those m categories with sufficient labeled training images are good and can be employed to perform predictions on unseen samples. However, the classifiers $\{\mathbf{w}_{m+1}, \ldots, \mathbf{w}_n\}$ for the $n - m$ categories with only k images may not be sufficient to perform reasonable predictions. Hence, the goal is to learn better classifiers for these $n - m$ categories.

A similar approach as that introduced in Section 11.4.1 can be used to refine the classifiers $\{\mathbf{w}_{m+1}, \ldots, \mathbf{w}_n\}$. Specifically, these learned classifiers $\{\mathbf{w}_{m+1}, \ldots, \mathbf{w}_n\}$ can be used as the input of a GNN model to produce the refined classifiers. The

GNN model can be trained on those categories with sufficient labels (Gidaris and Komodakis, 2019). Especially, to mimic the process of refining less-well trained classifiers to generate well-trained classifiers, for each of the categories with sufficient training samples, we can sample k training samples to form a "fake" training set, which simulates the setting of those categories with only k labeled samples. Then, a set of classifiers $\{\hat{\mathbf{w}}_1, \ldots, \hat{\mathbf{w}}_m\}$ can be learned from the "fake" training sets. These "fake" classifiers $\{\hat{\mathbf{w}}_1, \ldots, \hat{\mathbf{w}}_m\}$ and the ones learned with sufficient training samples $\{\mathbf{w}_{m+1}, \ldots, \mathbf{w}_n\}$ can be used as training data to train the GNN model. Specifically, the GNN model is similar to the one introduced in Section 11.4.1, where the difference is that, instead of using word embedding as input, the model now takes the "fake" classifiers as input. After training, the GNN model can be utilized to refine the classifiers $\{\mathbf{w}_{m+1}, \ldots, \mathbf{w}_n\}$ for those categories with k training samples. As mentioned in Section 11.4.1, knowledge graphs describing the relations between the categories can be used as the graphs, which the GNN model is applied to. In (Gidaris and Komodakis, 2019), the graph between the categories is built upon the similarity between the classifiers before refining.

11.4.3 Multi-label Image Classification

Given an image, the task of multi-label image classification is to predict a set of objects that are presented in the given image. A simple way is to treat this problem as a set of binary classification problems. Each binary classifier predicts whether a certain object is presented in the image or not. However, in the physical world, certain objects frequently occur together. For example, the *tennis ball* and the *tennis racket* frequently co-occur. Capturing the dependencies between the objects is key to the success of the multi-label image classification model. In (Chen et al., 2019c), a graph describing the relations between the objects is learned from the training set, and a graph neural network model is applied to this graph to learn inter-dependent classifiers. These classifiers predict whether objects are presented in a given image or not. Similar to that in Section 11.4.1, the classifiers are denoted by vectors $\mathbf{w}_i$.

Given an image I, the goal of multi-label image classification is to predict which objects from a candidate set $C = \{c_1, \ldots, c_K\}$ are presented in the given image. Hence, a set of K binary classifiers need to be learned to perform the task, which can be denoted as $\{\mathbf{w}_1, \ldots, \mathbf{w}_K\}$, with $\mathbf{w}_i \in \mathbb{R}^d$. The dimension d of the classifiers is defined by the image representation $\mathbf{x}_I \in \mathbb{R}^d$, which can be extracted by some pre-trained convolutional neural networks. To learn the object classifiers, which can capture the inter-dependencies between the objects, a graph neural network model is applied to a graph $\mathcal{G}$ which describes the relations between the objects. In (Chen et al., 2019c), the graph $\mathcal{G}$ consists of the objects as nodes and the connections between them are built according to their co-occurrence in the training data. Specifically, we first count the co-occurrence (i.e., appearing in the same image) of any pair of objects in the training set and get a matrix $\mathbf{M} \in \mathbb{R}^{K \times K}$ where $\mathbf{M}_{i,j}$ denotes the count of co-occurrence of i-th and j-th objects. Then, each row of this matrix is normalized as:

$$\mathbf{P}_i = \mathbf{M}_i / N_i,$$

where $\mathbf{P}_i$, $\mathbf{M}_i$ denote the i-th row of matrices $\mathbf{P}$, $\mathbf{M}$ respectively and N_i is the occurrence of the i-th object. To sparsify the matrix $\mathbf{P}$, we further use a threshold τ to filter the noisy edges as:

$$\boldsymbol{A}_{i,j} = \begin{cases} 0, & \text{if } \boldsymbol{P}_{i,j} < \tau; \\ 1, & \text{if } \boldsymbol{P}_{i,j} \geq \tau \end{cases}$$

The matrix $\mathbf{A}$ can be regarded as the adjacency matrix of the built graph. Once the graph is constructed, the graph neural network model can be applied to learn the classifiers for different objects. Specifically, the classifiers for the objects are the output of the graph neural network model, where the input is the word embeddings for these objects. After obtaining the classifiers $\{\mathbf{w}_1, \ldots, \mathbf{w}_K\}$, the classification can be done by mapping the image representation $\mathbf{x}_I$ to a score $\mathbf{w}_i^T \mathbf{x}_I$ that can be utilized for binary classification for each object c_i. Note that the entire process is end-to-end with the image as input and the prediction as output.

11.5 Point Cloud Learning

Point clouds provide flexible geometric representations for 3-D shapes and objects. More formally, a point cloud consists of a set of points $\{v_1, \ldots, v_n\}$ where each point contains 3-D geometric coordinates $v_i = (x_i, y_i, z_i)$ representing geometric locations. A point cloud can usually represent a 3-D object or shape. Like graphs, the point clouds are irregular as the points in the set are not ordered and not well-structured. Hence, it is not straightforward to apply classical deep learning techniques such as CNNs for point cloud learning. The topological information in the cloud points is implicitly represented by the distance between the points. To capture the local topology in a cloud point, a graph is built based on the distance between the set of points in the point cloud (Wang et al., 2019k). Specifically, k-nearest neighbors of each point v_i are considered as its neighbors in the built graph. Then, graph filtering operations are utilized to learn the representations for the points, which can be utilized for downstream tasks. Similar to graphs, there are two types of tasks on point clouds – one is point-focused task such as segmentation, which aims to assign a label for each point and the other is cloud-focused task such as classification, which is to assign a label for the entire point cloud. For the cloud-focused task, pooling methods are required to learn a representation from the point representations for the entire point cloud. Next, we describe the graph filtering operation introduced in (Wang et al., 2019k). For a single point v_i, the process can be expressed as:

$$\mathbf{F}_i^{(l)} = \text{AGGREGATE}\left(\left\{h_{\mathbf{\Theta}^{(l-1)}}(\mathbf{F}_i^{(l-1)}, \mathbf{F}_j^{(l-1)}) \mid v_j \in \mathcal{N}^{(l-1)}(v_i)\right\}\right), \tag{11.10}$$

where AGGREGATE() is an aggregation function such as summation or maximum as introduced in the GraphSAGE-Filter (see Section 5.3.2 for details of GraphSAGE-Filter), the function $h_{\mathbf{\Theta}^{(l-1)}}()$ parameterized by $\mathbf{\Theta}^{(l)}$ is to calculate the edge information to be aggregated. Various $h_{\mathbf{\Theta}^{(l-1)}}()$ functions can be adopted and some examples are listed below:

$$h_{\mathbf{\Theta}^{(l-1)}}(\mathbf{F}_i^{(l-1)}, \mathbf{F}_j^{(l-1)}) = \alpha(\mathbf{F}_j^{(l-1)}\mathbf{\Theta}^{(l-1)}), \tag{11.11}$$

$$h_{\mathbf{\Theta}^{(l-1)}}(\mathbf{F}_i^{(l-1)}, \mathbf{F}_j^{(l-1)}) = \alpha\left(\left(\mathbf{F}_j^{(l-1)} - \mathbf{F}_i^{(l-1)}\right)\mathbf{\Theta}^{(l-1)}\right), \tag{11.12}$$

where $\alpha()$ denotes a non-linear activation function. Note that in Eq. (11.10), $\mathcal{N}^{(l-1)}(v_i)$ denotes the set of neighbors of v_i, which is the k-nearest neighbors (including node v_i itself) calculated based on the output features $\mathbf{F}^{(l-1)}$ from the previous layer. Specifically, $\mathcal{N}^{(0)}(v_i)$ is calculated based on $\mathbf{F}^{(0)}$, which are the associated coordinates of the points. Hence, during training, the graph is evolving as the node features are updated.

11.6 Conclusion

This chapter introduces graph neural network models in various computer vision tasks, including visual question answering, skeleton-based human action recognition, zero-shot image recognition, few-shot image recognition, multi-label image recognition, and point cloud learning. For each task, we briefly introduce the task and describe why and how graph neural networks can improve its performance with representative algorithms.

11.7 Further Reading

In addition to the computer vision tasks we introduced in this chapter, graph neural networks have been adopted to enhance many other tasks in computer vision. In (Ling et al., 2019), it is utilized to annotate objects from given images. Graph neural networks are adopted to deal with scene graphs and improve the performance of many tasks related to scene graphs, including scene graph generation (Chen et al., 2019a; Khademi and Schulte, 2020), and scene graph based image captioning (Yang et al., 2019).