**Segmentation Using Graph-Based Clustering**

使用圖論分群的影像分割技術

游承軒 著 丁建均老師 修訂

國立台灣大學電信工程學研究所

2021.12

目錄 (Contents)

1. Introduction to Graph-Based Clustering 3
2. Graph-Based Segmentation Methods 8
   1. Normalized Cuts 9
   2. Efficient Graph-Based Image Segmentation (FH) 13
   3. Segmentation by Aggregating Superpixels (SAS) 16
   4. Entropy Rate Superpixel Segmentation (ERS) 20
   5. A Global/Local Affinity Graph for Image Segmentation (GL-graph) 24
   6. Superpixel Segmentation using Linear Spectral Clustering (LSC) 29
   7. Graph-Without-Cut: An Ideal Graph Learning for Image Segmentation 34
   8. Other Conventional Graph-based Segmentation Methods 36
3. Graph-Based Segmentation Methods Integrating Deep Learning 37
   1. Classic CNN-Based Segmentation Methods 39
      1. Fully Convolutional Networks for Semantic Segmentation (FCN) 39
      2. Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation (DeepLabV3+) 40
   2. Beyond Grids: Learning Graph Representations for Visual Recognition(GCU) 41
   3. Graph-Based Global Reasoning Networks (GloRe) 45
   4. Spatial Pyramid-Based Graph Reasoning for Semantic Segmentation (SpyGR) 48
   5. Other Graph-based Segmentation Methods Using Deep Learning 50
4. Conclusion and Further Reading 51
5. Reference 52

**1. Introduction to Graph-Based Clustering**

In the graph theory, a graph is a structure amounting to a set of objects in which some pairs of the objects are; "related". Each object is called a "vertex" and each related pair of two objects is called "edge". Mathematically, a graph G = (V, E), where V and E denotes the sets of vertices (also called nodes) and edges, and use || to denote the cardinality (which means the number of elements) of a set. In practice, the adjacency matrix A of G can explicitly characterize the structure of the graph, so obtaining A is very important. We often compute the degree matrix D= and L = D – A, because the Laplacian matrix (graph Laplacian) L has beautiful mathematical properties and gives us a closer look to the graph.

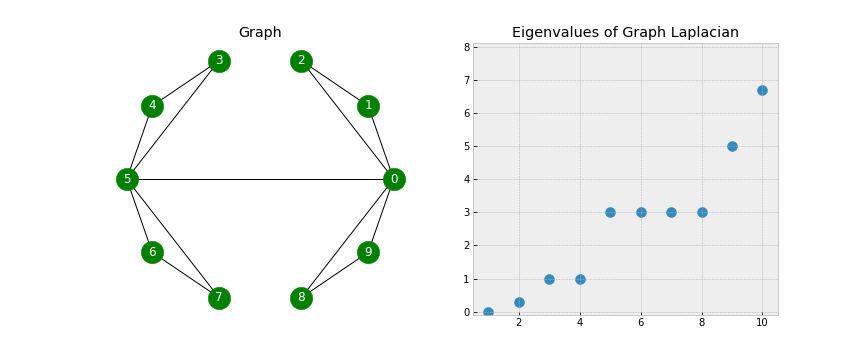


Fig. 1.1. A graph and its eigenvalues (from small to large) of the Laplacian matrix. [11]

In Fig. 1.1, the left subfigure shows a graph structure. The corresponding A, D, and L matrices of the graph is written as follows:



To analyze the structure of a graph, we often perform spectral decomposition on the Laplacian matrix L to acquire its eigenvectors and eigenvalues. The eigenvalues and its corresponding eigenvectors of L derived from the left subfigure of Fig. 1 are written as follows:

where is a small enough positive/negative number near to 0. Eigenvalues are located in diagonal entries, and each eigenvalue corresponds to eigenvectors on each column. The magnitude of eigenvalues is illustrated in the right subfigure of Fig. 1. We can observe that the first eigenvalue is 0, and the second eigenvalue is near to 0 because there is almost a separation of two components by cutting edge (5, 0). In mathematics, the number of zero-value eigenvalues reveals the number of connected components in a graph, and the second smallest eigenvalue is called the Fiedler value, and its corresponding eigenvector is called the Fiedler vector. We can simply separate nodes based on whether its corresponding entries in the Fiedler vector are positive or not. In this example, the Fiedler vector is the column vector with red colors in EigVec. Fig. 1.2 shows the partition result.

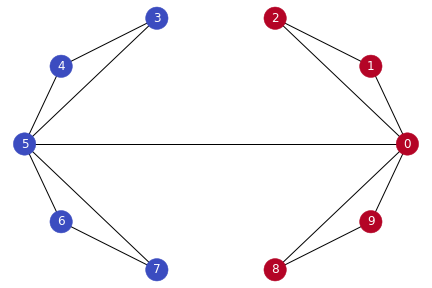


Fig. 1.2. The segmentation of the graph with the same color as a group. [11]

The Fiedler vector (i.e. the eigenvector with the second smallest eigenvalue) gives the information about the grouping of each node, so we can utilize this property from the Laplacian matrix to partition the graph based on the sign of entries in the Fiedler vector. If we want to partition the notes into *k* groups simultaneously, we can keep *k* smallest eigenvectors of a Laplacian matrix, where we treat each eigenvector as a unique feature of a node. Performing K-means clustering with the number of clusters = *k* gives the simultaneously *k*-way partition of the graph. Take the graph in Fig. 1.1 for example again. If we want to partition the graph into 4 groups, we use the smallest 4 eigenvectors as 4 unique features of each node:

where each row represents a node with 4 features. After performing K-means on the 4-dim feature space, we can separate the graph into 4 groups. Fig. 1.3 shows the grouping result of the graph.

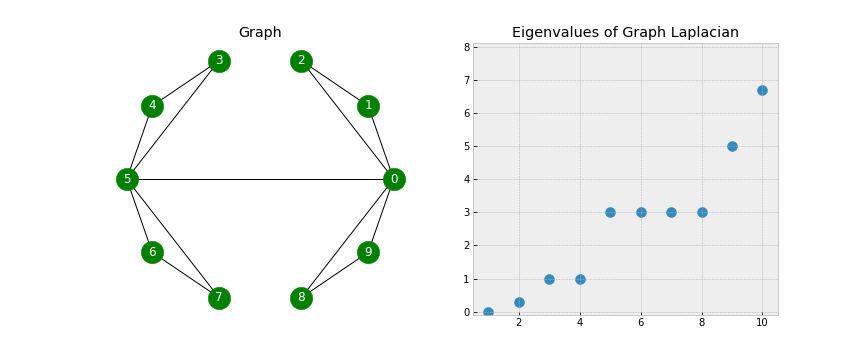
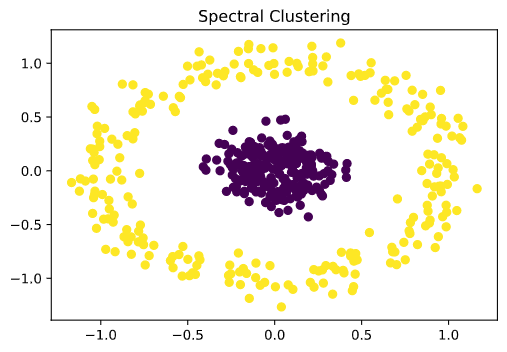
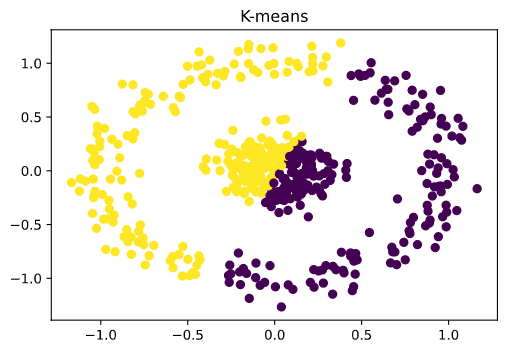


Fig. 1.3. The grouping result of the graph, with each group circled by an ellipse.

If the data is not a graph-like structure, one can construct edges between data points by defining weights as similarities between nodes to make the original data form a graph. Usually, we obtain weights by computing pairwise feature differences of data points. A common definition is , where denotes features of the kth data point, and controls the rate of weight decay of distant nodes.



(a) (b)

Fig. 1.4. Data points clustered by (a) K-means and (b) graph-based (spectral) clustering.

Take another 2-D circle-distributed data for example. Fig. 1.4 shows clustering by standard K-means algorithm with cluster number *k* = 2 and the graph theory. As one can see, the performance of the graph-based clustering method is better than the one of the K-means for its ability to partition data with non-convex boundaries. This is due to that, in the K-means algorithm, the nodes are clustered according to the distances to the cluster centers and in the graph-based method the nodes are clustered according to the relations among nodes.

The two examples give us an intuitive to graph-based segmentation. The idea can be applied to image segmentation. In Chapter 2, we will introduce conventional graph-based image segmentation methods. In Chapter 3, we will give a connection between the graph-based clustering and the deep learning, and discuss several related deep learning-based methods.

**2. Graph-Based Segmentation Methods**

Traditional graph-based segmentation methods usually treat an image as a weighted undirected graph with nodes. Each node is a pixel or a superpixel in an image. Weights between edges can be defined as similarities or dissimilarities. After nodes are decided and weights are computed, different ways can be used to partition or merge the graph into *k* mutually excluded groups (i.e. subgraphs, components, parts, clusters, segments, regions) with no edge connecting each other. Each group corresponds to a segmented region. Based on parameter settings, some methods [1, 5] can also be used to generate superpixels, while some other methods [12, 16] are specific to superpixel generation.

Table 2.1. Summary of the methods discussed in Chapter 2.

|  |  |  |  |
| --- | --- | --- | --- |
| Methods | Mainly-Used Technique | Advantage | Procedure |
| Normalized Cuts [1] | Generalized eigen system | Mathematically solid foundations | Cut into regions |
| FH [5] | (1) Minimum spanning tree  (2) Greedy algorithm | Fast (union find structure) | Merge into regions |
| SAS [7] | (1) Sparse bipartite graph  (2) Generalized eigen system | (1) Fast (sparsity)  (2) Respect object boundaries  (3) Capture multiscale information | 1. Over-segment  2. Merge into regions |
| ERS [12]  (For superpixels) | (1) Entropy  (2) Greedy algorithm | (1) Fast  (2) Respect object boundaries | Merge into regions |
| GL-Graph [13] | (1) *L*0-norm sparse graph  (2) Sparse bipartite graph  (3) Generalized eigen system | (1) Fast (sparsity)  (2) Respect object boundaries  (3) Capture multiscale information | 1. Over-segment  2. Merge into regions |
| LSC [16]  (For superpixels) | (1) Weighted K-means  (2) Kernel function | (1) Fast (K-means)  (2) Respect object boundaries | Cut into regions |
| GWC [18] | Iterative convex optimization | (no code)  (1) Fast (convex solver)  (2) Respect object boundaries | 1. Over-segment  2. Merge into regions |

**2.1 Normalized Cuts**

The Normalized Cut [1] treats image segmentation as a graph partitioning problem. It applies a global criterion, called the normalized cut, for segmenting the graph. It measures both the total dissimilarity between different groups as well as the total similarity within the groups. A cut is to partition nodes into several groups and a group is a set of nodes. The Normalized Cut finds the optimal segmentation by minimizing the measurement.

The dissociation measure:



(2.1)

where  is the total weight of the edges between two groups *A* and *B*. It measures the degree of dissimilarity. For example, Fig. 2.1 is a typical graph structure. The red-dotted line sums the weight along its trajectories and segments the graph into two groups A and B, which minimizes *cut*(*A*, *B*). For the red-dotted line in Fig. 2.1, *cut*(*A*, *B*) = 9.

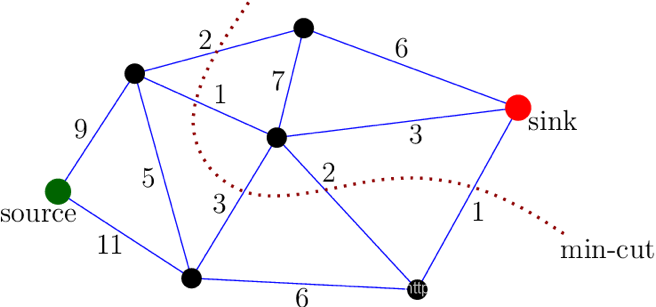


Fig. 2.1. An example of minimum cut partition. [3]

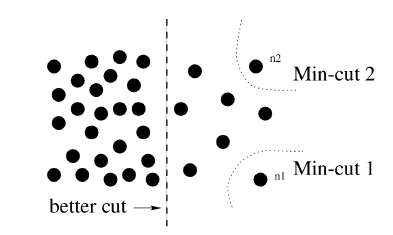


Fig. 2.2. A case where the minimum cut gives a bad partition. [1]

However, if we only minimizes *cut*(*A*, *B*), the algorithm tends to segment small regions as shown in Fig. 2.2 because it does not consider the size of each group (segment). Different from minimizing the cut, *Ncut*(*A*, *B*) prevents from segmenting small regions.  is the total weights from nodes in the group A to all nodes (denoted by *V*) in the graph. In the case of segmenting small regions, *assoc*(*A*, *V*) will be small, and *Ncut*(*A*, *B*) will be relatively large, which means that *Ncut*(*A*, *B*) has not been optimized.

If we want to partition the nodes into two sets A and B, let *x* denotes an *N* dimensional vector where *N* is the number of nodes, with *xi* =1 if node *i* is in the group A and -1 otherwise. Let



be the total weights from node i to all other nodes. Then, (2.1) can be rewritten as:



(2.2)

The solution of *x* that can minimize (2.2) is equivalent to the solution to minimize the Rayleigh quotient [2]:



(2.3)

where , *D* is an *N* by *N* matrix with *di* on its diagonal, and *W* is an *N* by *N* symmetric matrix with . Eq. (2.3) was proved in [1].

However, the problem is NP-complete. If *y* is constrained to be real, it can be solved using the generalized eigenvalue system:

.

(2.4)

where *λ* and *y* represent the eigenvalue and the eigenvector. Taking the second smallest eigenvectorof *y* gives the solution to (2.4) by assigning node *i* to the group A if the ith entry in the second smallest eigenvector is larger than 0 and otherwise.

**Summary of the Normalized Cut Algorithm**

(1) Given an image or an image sequence, construct a weighted graph G = (V, E) and set the weight W on the edge connecting two nodes to be a measure of the similarity between the two nodes. Then compute the diagonal matrix **D**.

(2) Solve (**D**-**W**)*y* = *λ***D***y*.

(3) Use the eigenvector with the second smallest eigenvalue to bipartition the graph.

(4) Decide if the current partition should be subdivided and recursively repartition the segmented parts if necessary.

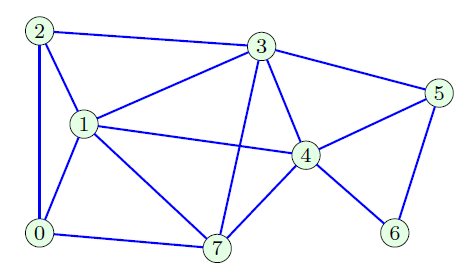


Fig. 2.3. An undirected graph. [4]

The figure above shows a toy example. Assuming weight=1 for all edges, we can construct an adjacency matrix **A** and compute **D**:

Then **L**=**D**-**A**, and solve the generalized eigen-problem to obtain eigenvectors **EigVec**. We only keep the first and the second eigenvectors, and we can observe that entry 4, 5, 6 in the second smallest eigenvector > 0, which implies us to partition the graph along the black line. Node 4, 5, 6 belong to the same group and other nodes belong to the other one.

**2.2 Efficient Graph-Based Image Segmentation (FH)**

Different from the Normalized Cuts [1], Felzenszwalb and Huttenlocher proposed a graph-based method, denoted as FH [5], where edges are defined as dissimilarities between pixels. It is a method based on minimum spanning tree (MST). A spanning tree T of a graph G=(V, E) is a tree containing all nodes in the graph while edges are a subset of E. A graph may have several different spanning trees. The MST is a spanning tree with the smallest total edge weights among all spanning trees. The FH algorithm performs segmentation initially with each pixel being a connected component (no edges connected), and gradually merges nodes into components by adding edges with a defined adaptive threshold.

The merging threshold to decide whether two MST groups C1 and C2 should be merged is defined as follows:



(2.5)

where |*et*| is the smallest edge weight connecting MST C1 and MST C2, meaning the differences across the two MSTs. Int(C) is the largest edge weight in the MST C, meaning the internal difference inside a subgraph/MST. |*C*| denotes the number of nodes of *C*, and K is some constant. The equation means that if the smallest edge weight (dissimilarity) connecting two MSTs is smaller than the largest weight in each MST plus a number, then merge the two MSTs.  allows two sufficiently small components to be merged.

Different from a constant merging threshold, the FH algorithm adopts an adaptive merging threshold. For example, a smooth region usually has similar pixel values, which means that the internal difference Int() is small in its MST, which implies the region can only merge pixels with “very” similar colors. In contrast, a region with high variability has large Int(C), which implies the region can merge pixels with “not quite” similar colors. In other words, the merging criteria is stricter for smooth regions, while it will be loose for rugged regions.

**Summary of the FH Algorithm**

The input is a graph G = (V, E), with *n* vertices and *m* edges. The output is a segmentation of V into components S

(1) Sort *m* edges of E in a non-decreasing order

(2) Start with a segmentation *S*(0), where each vertex vi is in its own component. Initialize *q* by 1.

(3) Let vi and vj denote the vertices connected by q-th edge in the non-decreasing ordering, i.e., oq = (vi, vj). If vi and vj belong to different groups of *S*(*q*−1) (*S*(*q*−1) means the segmentation results at the (*q*−1)th iteration) and the weight of *oqw*(*oq*) is small compared to the internal difference of both those groups, then merge the two components. Otherwise, nothing is done. Formally, this step examines whether equation (2.5) is true.

(4) Repeat step (3) for *q* = 1, …, *m*.

(5) Output the finial segmentation result S = S(m).

In Fig. 2.5, a segmentation result using the FH algorithm is shown. In Fig. 2.5, the weights are defined as follows:



(2.6)

where *I*(*pi*) is the intensity of the pixel *pi*.

**Results**

Fig. 2.5. A duck, and the segmentation results produced by FH (σ = 0.8, k = 300). Pixels are 4-connected, as shown in Fig. 2.6.

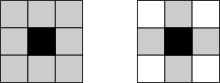


Fig. 2.6. Example of neighborhood of pixels - association of four pixels [6].

**2.3 Segmentation by Aggregating Superpixels (SAS)**

Li *et al.* proposed a novel segmentation framework, SAS [7], based on bipartite graph partitioning, which treats superpixels as nodes and aggregates them by different algorithms to form a bipartite graph structure.

**Graph Construction**

Denoting S as a set of multi-layer superpixels over an image I, a bipartite graph is a graph whose vertices can be divided into two disjoint and independent sets X and Y such that every edge connects a vertex in X to one in Y. In this method, X is composed of pixels and multi-layer superpixels produced by different superpixel algorithms, with number of nodes *NX* = |I|+|S| (|I| means the number of pixels and |S| means the number of superpixels), while Y is composed of multi-layer superpixels, with number of nodes *NY* = |S|. We construct edges connecting nodes in X and nodes in Y in two ways:

(1) superpixel cues: connect pixels to a superpixel

(2) smoothness cues: connect neighboring superpixels that are close in the feature space

The cross-affinity matrix (weight matrix) B =  is defined as follows:



(2.7)

where  are parameters, *dij* is the distance between the features of xi and yj, xi and yj are feature vectors of nodes in X and Y and the tilde ~ means neighborhood between superpixels.

**Graph Partition**

For an ordinary graph, we can use spectral clustering to partition it into *k* groups, which solves the generalized eigenvalue problem as introduced in Chapter 1:



(2.8)

where L is the graph Laplacian matrix and D is the degree matrix of the graph. In [7], Li tried to reduce computation by proving that if a bipartite graph is highly sparse, the following formula can be derived from the original eigen-problem in Eq. (2.8):



(2.9)

where LY = DY – WY, DY = diag(BT**1**), DX = diag(B**1**) and WY = BTDX-1B. LY is the Laplacian of the graph GY = {Y, WY}, DY and DX are the degree matrices of X and Y, respectively [8, 9], and B is defined in Eq. (2.7). Li *et al.* [7] stated that the smallest *k* eigenvectors of L in Eq. (2.8) can be obtained from the smallest *k* eigenvectors of LY in Eq. (2.9) which can be computed efficiently because GY corresponds to the much smaller graph over superpixels () and thus reduces complexities.

Take a 500\*500 image for example. In the bipartite graph, 5 different superpixel segmentations are used, and a pixel is connected to each superpixel if its location is within the superpixel. The 5 over-segmentations have 600 superpixels totally. B = , and the number of nonzero entries is 250000\*5+600\*600. Since only about 1% of entries in B are nonempty, so the graph is highly sparse.

**The Algorithm**

Input: An image I and the number of segments *k*.

Output: A *k*-way segmentation of I.

(1) Collecting a bag of superpixels S for I by different methods (e.g., the Mean Shift superpixels [10]. The result of the FH algorithm [5] can also be treated as superpixels)

(2) Construct a bipartite graph G = {X, Y, B} with X = I∪S, Y = S, and B defined in (2.7)

(3) Apply the Transfer Cuts (Tcut) algorithm introduced below to derive *k* groups of G, and treat each group as a segment

**Transfer Cuts**

Input: A bipartite graph G = {X, Y, B} and *k* (the number of groups of the segmentation result).

Output: Segmentation results of G

(1) Form DY = diag(BT**1**), DX = diag(B**1**), WY = BTDX-1B and LY = DY – WY

(2) Compute the smallest *k* eigenpairs  of 

(3) Obtain  such that  and , i=1,…, k

(4) Compute , with , *i* =1, 2, …, *k*.

(5) Perform K-means to derive *k* groups of X∪Y from f1,…,fk., and record the cluster number of each pixel.

(6) Segment the image by the *k* partition of pixels.

**Results**

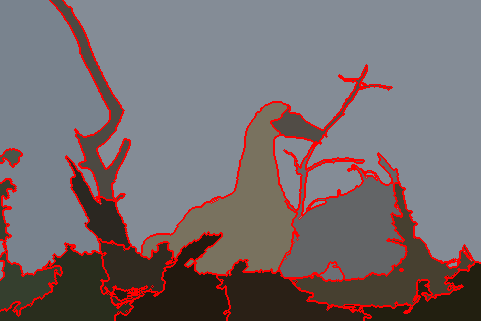
 

Fig. 2.7. Segmentation by SAS algorithm with *k*=16. We find that SAS performs well on the object boundaries as well as the background. Among the non-learning based segmentation algorithm, the SAS algorithm is one of the algorithms that have the best performance.

**2.4 Entropy Rate Superpixel Segmentation (ERS)**

Liu *et al.* proposed the entropy rate superpixel (ERS) segmentation algorithm [12]. They treated superpixel-based segmentation as a clustering problem and proposed a new clustering objective function for superpixel segmentation. This objective function to generate superpixels consists of two components:

(i) the entropy rate of a random walk on a graph and (ii) a balancing term.

The entropy rate favors compact and homogeneous clusters which are perceptually consistent and each of them almost contains only a single object; whereas the balancing term balances the size of each region of the segmentation result.

**Preliminaries**

**Entropy:** The uncertainty of a random variable is measured by entropy *H*. The entropy of a discrete random variable X with a probability mass function *pX* is defined by



(2.10)

where *χ* denotes possible outcomes of X.

**Entropy rate:** The entropy rate quantifies the uncertainty of a stochastic process **X** = {*X*t|*t* ∈ *T*} where *T* is some index set. For a discrete random process, the entropy rate is defined as an asymptotic measure



(2.11)

which measures the uncertainty of the random process *Xt* after observing the past trajectory {*Xn*|*n* = 1, 2, …, *t* −1}.

**Random walks on graphs:**

Let *X* = {*Xt*|*t* ∈ *T*, *Xt* ∈ *V*} be a random walk on the graph G = (V, E) with a nonnegative similarity measure *w*. The transition probability is defined as:



(2.12)

where  is the sum of the weights on the edges out from the vertex , and the stationary distribution is given by:



(2.13)

where  is the normalization constant.

**Problem Formulation and the Objective Function**

We want to select a subset of edges A ⊆ E such that the resulting graph, G = (V, A), contains exactly *K* connected subgraphs by using the entropy rate of the random walk on the constructed graph and the balancing term. The entropy rate of the random walk on G = (V, A) is defined as:



(2.14)

where  and  are defined in (2.13) and (2.12), respectively. The balancing term is defined as:



(2.15)

where *NA* is the number of connected components in the graph and  representing the distribution of cluster membership in which |Si| is the number of nodes of the *i*th group.

If the pixels in a superpixel are consistent (i.e., the differences of pixels are small within the superpixels), then the weights in a superpixel are similar, which makes  and  large and leads to a larger *H*(*A*) in (2.14). On the other hand, if the number of pixels in a superpixel is similar across each superpixel, then  is almost the same for each superpixel, which leads to a larger *B*(*A*) by (2.15). Thus, maximizing *H*(*A*) and *B*(*A*) can achieve the goal of grouping perceptually consistent regions while keeping similar sizes for each superpixel.

The clustering is achieved via optimizing the objective function with respect to the edge set:



subject to  

(2.16)

where *λ* ≧0 controls the importance of the balancing term. *K* denotes the number of clusters. The additional constraint on the number of connected subgraphs enforces exactly *K* clusters since the objective function is monotonically increasing under the graph construction (number of clusters *K* non-increasing with edges added).

The objective function can be optimized via greedy heuristic, which starts with an empty set (A = {∅}) and at each iteration, add the edge that yields the largest gain of the objective function. The iterations are stopped when the number of connected subgraphs reaches a preset number, *NA* ≤ *K*. Detailed procedures are described as the following algorithm.

**The ERS Segmentation Algorithm**

(1) Map an image to a grid graph (8-connected) G = (V, E) with similarities

*wi*, *j* = 

where *d*(*vi*, *vj*) = intensity difference × spatial difference between pixels.

(2) Initial Setting: G = (V, E), *wi*, *j* is defined as in step (1), *K* is the number of clusters, and *λ* is a constant, A←∅, U←E, and the loss function is.  denotes an edge in the edge set U.

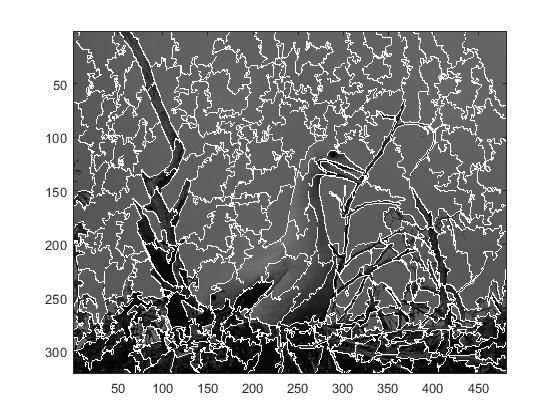
(3) Repeat



until U=∅.

(4) Segmentation result: A.

**Results**

(a) original image (b) ERS superpixels

Fig. 2.8. The superpixels generated by the ERS method with *K* = 200. The superpixels (boundaries shown in white) respect object boundaries and tend to divide an image into regions with similar size and irregular shape.

**2.5 A Global/Local Affinity Graph for Image Segmentation (GL-graph)**

Many graph-based segmentation methods perform well on local patterns of an image because they consider local relationships between pixels by adjacency graphs. However, they cannot capture global information. In other words, they fail to group distant pixels into the same regions and tend to separate them.

Wang *et al.* proposed a novel sparse global/local affinity graph [13] over the superpixels of an input image to capture both short- and long-range information, and segment it through a suitable graph-cut algorithm. This is achieved by over-segmenting the input image into superpixels at different scales as described in SAS [7], implementing a law to define small, medium and large sized superpixels which makes use of small and large sized superpixels to construct adjacency graphs, and encoding local relationship. Medium sized superpixels are applied to construct an *L*0-sparse graph to capture long-range information. A bipartite graph is also introduced in [7] to enable the propagation of information across the superpixels at different scales. Moreover, the partition is performed efficiently through the Transfer Cuts algorithm [7] (also see Section 2.3).

**Process**

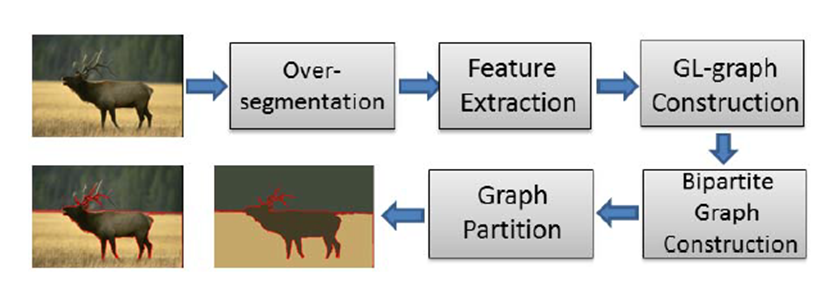


Fig. 2.9. The flowchart of the GL-graph algorithm. [13]

(1) Multi-Scaled Superpixel Generation and Representation

Superpixels generated by different methods with varying parameters can capture various and multiscaled visual patterns of a natural scene image. For each superpixel, three perceptual features (color, texture, and shape) are considered.

(2) Global/Local Affinity Graph Construction

Small-sized superpixels are the superpixels whose area is smaller than an area threshold. To determine which superpixels are large-sized ones, we sort all superpixels of a scale in an ascending order by areas, compute cumulative sums of superpixel areas, draw a curve with x-axis being the number of superpixels and y-axis being cumulative sums, and find the largest second derivative of the curve with its corresponding area being selected as threshold of large-sized superpixels. Fig. 2.10 illustrates the result. Medium-sized superpixels are remaining unselected superpixels.

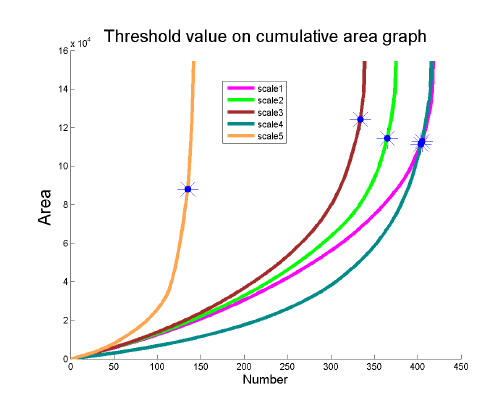


Fig. 2.10. Illustration of selecting the area threshold for large-sized superpixels.

(i) Building an L0-Graph for Medium-Sized Superpixels:

First, approximate medium-sized superpixels in a given feature space as a linear combination of other medium-sized superpixels of the same image in a feature space:



(2.17)

where || ||0 denotes the *L*0-norm of the vector, which means the number of nonzero entries of a vector, Y is the feature matrix of superpixels, with each column yi being the feature vector of the *i*th superpixel: . *ci* is the coefficient vector, where each entry in *ci* represents a coefficient, and *cii* is the *i*th entry. **Y***ci* is the linear combination of each feature vector in **Y**, *cii* = 0 for preventing multiplication with the *i*th column (*yi*) itself. The equation means that it aims to reconstruct a feature vector by other feature vectors with the minimum number of vectors.

However, it is NP-hard to find the sparsest solution of linear equations. Instead, Wang *et al.* [13] used the orthogonal matching pursuit (OMP) [14] to approximate the solution. The OMP solves the L0-norm sparse representation of the formula by the following optimization problem:



(2.18)

where || ||2 denotes *L*2-norm and *L* controls the sparsity of *ci*.

After obtaining *ci* for each superpixel, we can define the reconstruction error of the feature of the *i*th superpixel represented by other superpixels:



(2.19)

where *cij* is the *j*th entry in *ci*. The similarity coefficient *wij* of the similarity matrix *W* between superpixels *si*, sj is defined as:



(20)

(ii) Building an Adjacency-Graph for Small and Large Sized Superpixels with Respect to Their Neighbors:

Traditionally, the pairwise similarities are computed with the Gaussian kernel function with standard deviation denoted as σ. It is hard to decide adaptively the value of σ in order to maintain the same order of magnitude with the *L*0-graph. Therefore, Wang adopted the same way as for the *L*0-graph to compute the similarities described in the next paragraph.

Given a superpixel si with its corresponding feature vector xi and the matrix-representation **D** with each column being a feature vector of its adjacent superpixel neighbors in the image, we try to represent xi as a linear combination of columns in **D**. In practice, we solve the following optimization problem:



(2.21)

where ci is an analogue of ci in (2.17). Once has been computed, the similarities between a superpixel and its neighbors in the image are computed as in (2.19) and (2.20).

(3) Fusing GL-Graphs of Different Visual Features and Different Scales

After constructing similarities for features, we implement a simple weighted sum of three major visual features (color, texture, and shape) to fuse these similarities into a single affinity matrix:



(2.22)

where *m* denotes the number of visual features and *β* is the weight for each type of feature.

Multi-scale superpixels mentioned in SAS [7] are also adopted in GL-graph. To fuse *l* different superpixels, we plug the affinity matrix *W*l of each scale corresponding to its GL-graph into a block diagonal multiscale affinity matrix Wss:



(2.23)

This multiscale affinity matrix of superpixels preserves similarities for each scale.

(4) Bipartite Graph Construction and Partition

Build a bipartite graph G=(X, Y, B) with X and Y as two sets of nodes and B as cross-affinity matrix which consists of two parts describing the pixel-superpixel and superpixel-superpixel relationships respectively:



(2.24)

where  if pixel i belongs to superpixel j,  otherwise.  as defined in (2.23). |I| and |V| denote the number of pixels and multiscale superpixels. The bipartite graph is sparse, so we can apply Transfer Cuts in SAS [7] to efficiently partition the graph. The process and the concept of (4) is the same as described in Section 2.3.

**Results**

(a) Original image (b) Segmentation result

Fig. 2.11. GL-graph segmentation with number of clusters *k* set to 40. We find that the result adheres well to boundaries and it avoids dividing the sky region in the image despite the relatively large *k*.

**2.6 Superpixel Segmentation using Linear Spectral Clustering (LSC)**

LSC [16] is a clustering algorithm for generating compact and uniform superpixels with low computational costs. It is based on optimizing the objective function of weighted K-means in an iterative manner, which is similar to SLIC [17]. Instead of using the eigen-based algorithm such as Normalized Cuts [1], whose weight matrices are highly dense, Li *et al.* [16] proved the equivalence of optimizing the objective function of Normalized Cuts and weighted K-means by introducing an elaborately designed high dimensional space. Therefore, it can be solved in a simpler and more efficient way to save time and the memory cost.

To derive the equivalence, first, reexamine the objective function of 2-way partition of Normalized Cuts:



(2.25)

where  and , which are the same as described in Section 2.1. Eq. (2.25) can be rewritten as follows:



(2.26)

where *Nassoc*(*A*, *B*) measures the total normalized associations within each group. We can extend *Nassoc* into *k*-group partition:



(2.27)

The definition of Eq. (2.27) matches the following equation:

 (2.28)

where *p* and *q* are two pixels, *W*(*p*, *q*) is the similarity between them, denotes the *k*th cluster (group), *K* is the number of clusters, and V denotes the set of all clusters. Thus, minimizing the objective function of Normalized Cuts *Ncut* is equivalent to maximizing FNcuts defined in Eq. (2.28). The objective function of weighted *K*-means is defined as:

 (2.29)

where || || denotes the *L*2-norm, *w*(*p*) is the weight value defined in Eq. (2.31) and assigned to data point (pixel) *p*,  is the kernel function mapping *p* to a higher dimensional feature space to improve linear separability, and *mk* is the center of in the feature space.

Li *et al.* [16] proposed that optimizing objective functions of weighted *K*-means *Fk*-*m* and Normalized Cuts Ncuts is mathematically equivalent if Eq. (2.30) and Eq. (2.31) hold:

 (2.30)

 (2.31)

where V denotes the set of nodes (pixels). Eq. (2.30) indicates that the similarity between pixel *p* and *q* can be computed by the weighted inner product in the high dimensional feature space, and Eq. (2.31) indicates that the weight of a pixel is the sum of similarities between it and all other pixels. The proof of the two equations is in [16].

The equivalence of minimizing Fk-m and maximizing FNcuts is proved as follows:.



(2.32)

where  is a constant independent of clustering results because the double summation includes all pixels in an image.

So far, the equivalence of minimizing Ncuts in [1] and maximizing FNcuts and minimizing Fk-m are proved by Eq. (2.26) - Eq. (2.32). In LSC, w(p) is computed by Eq. (2.31) with ,  defined as follows to satisfy Eq. (2.30):



(2.33)



(2.34)

where x, y, l, α, β denote positions in coordinate space and lab values in the CIELAB color space. Cs and Cc are parameters controlling the relative importance of spatial and color feature, respectively. The author used a custom similarity definition  first, and defined others based on  to meet Eq. (2.30) and Eq. (2.31).

**The Linear Spectral Clustering (LSC) Algorithm**

**Input**: image, the value of K (i.e., the number of superpixels)

**Output**: K superpixels

(1) Map each pixel *p* to a ten-dimensional vector by ϕ(p) defined in (2.34) according to its x, y, l, α, β values

(2) Choose *K* different pixels within the image uniformly.

(3) Move each of the *K* points along the lowest gradient directions in their corresponding 3×3 neighborhood

(4) Initialize center *mk* in the ten-dimensional space by (2.29) and center *ck* in the (*x*, y) space respectively using *K* points in step (3)

(5) Set label *L*(*p*) = 0 and distance *D*(*p*) = for each pixel *p*.

(6) repeat

for each weighted means *mk* and search the center *ck*

do

for point p in the 2*vx* × 2*vy* neighborhood of *ck* in the image plane do

*Dk* = Euclidean distance between *ϕ*(*p*) and *mk* in the feature space.

if *Dk* < *d*(*p*) then

*d*(*p*) = *Dk*

*L*(*p)* = *k*

end if

end for

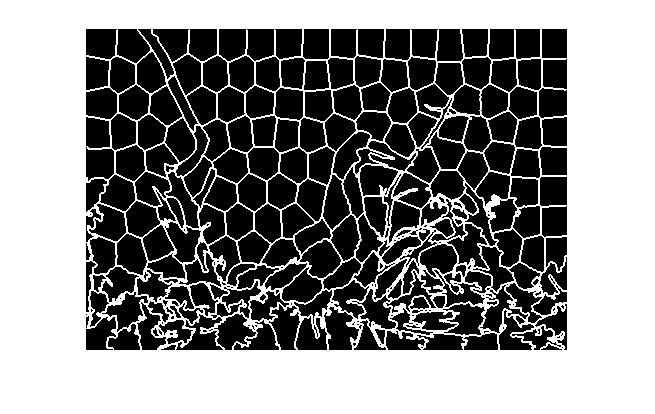
end for

Update weighted means and search centers for all clusters.

until all mk converge

(7) Merge small superpixels to their neighbors

Results

(a) Original image (b) Segmentation

Fig. 2.12. LSC with K = 200. We see that the superpixels generated by LSC match object boundaries and preserve global image structure.

**2.7 Graph-Without-Cut: An Ideal Graph Learning for Image Segmentation**

Most graph-based image segmentation methods first generate the similarity matrix for graphs, and then merge/cut the graph to make the final segmentation result have only K connected components. This two-step process may not lead to the optimal solutions, because it is hard to jointly optimize the two-step process. Gao *et al.* thus proposed a novel framework, which is called Graph-Without-Cut (GWC) [18], for learning similarities for graph and cutting it simultaneously. In short, GWC first generates superpixels. It then finds the similarities between superpixels and segment it into K regions at the same time by iterative optimization.

**Ideal Graph Learning**

Suppose that we extract *T* features of all *N* superpixels generated by other methods. Let  be the center location for the *N* superpixels, and  be the *t*th feature matrix of *N* superpixels with *t* = {1, …, *T*}. The final similarity matrix *S* of the superpixel graph should be inversely proportional to spatial and feature differences. Therefore, the objective function can be formulated as:



(2.35)

where g(Y, S) and h(Xt, S) are the loss functions to measure the smoothness of the graph S on the spatial location Y and the feature vector Xt. In (2.35), *r*(*S*, α) is the regularization term. They are defined as:



(2.36)



(2.37)



(2.38)

where || ||2 denotes the *L*2-norm while || ||F denotes the Frobenius norm. sij is the similarity between superpixels *i* and *j*, and *xi*, *yi* are the *i*th columns in X, and Y, respectively. *α* is a parameter vector with each entry controlling the importance of each feature, and μ, β in Eq. (2.35) andγ in Eq. (2.38) are also controlling parameters.

For a nonnegative similarity matrix S, there is a Laplacian matrix L = D – S, D being the diagonal matrix with nonzero values as the sum of S along one direction. L has the following property: The number of eigenvalues 0 of L is equal to the number of connected components of G with similarity matrix S. So if rank(L) = N − K, then the superpixels have K connected components based on S. Thus, the ideal graph learning model is to solve the optimization problem:



(2.39)

where constraints on S and α force them to be element-wise nonnegative and row-wise summation equal to 1.

The objective function means that it can learn the similarities between superpixels by making their pairwise position and color differences smaller, because the similarity matrix *S* is what it wants to optimize. Moreover, the rank constraint forces the similarities to be updated properly, which segments the image to a given number of regions finally. Therefore, GWC can learn similarities and perform segmentations simultaneously.

**2.8 Other Conventional Graph-based Segmentation Methods**

There are many other image segmentation methods based on the graph-based clustering theory.

The parametric kernel graph cut method (KM) [20] uses an objective function composed of a data term, which is a loss function to measure fitness of image data to assumed region parameters (e.g. the cluster centers) in a kernel-mapped space and a smoothness term, which is a loss function to measure the differences between the current and surrounding superpixels. This objective function not only considers the cluster centers to fit image data but also make the superpixels in the same cluster as similar as possible. The optimization process of objectives is conducted in a two-step iterative process: (1) update cluster centers by [21] with fixed partition and (2) update partitions by [22] with fixed cluster centers, which uses mincut to optimize.

The *L*0-graph [23] method uses *L*0-norm sparse representations to obtain weights (similarities) between multiscale superpixels. It is a simplified version proposed before GL-graph [13]. First, the representing coefficients is determined by solving an *L*0-norm optimization problem with the orthogonal matching pursuit (OMP) [14] and computing the weights between superpixels based on representing coefficients. Then, a bipartite graph as in SAS [7] is constructed and Transfer Cut is applied to partition the sparse bipartite graph.

**3. Graph-Based Segmentation Methods Integrating Deep Learning**

Methods discussed in Section 2 are all unsupervised. They treat image segmentation as a clustering problem and do not require labeled training images to optimize the methods. In contrast, deep learning-based segmentation methods are supervised learning, which requires lots of training images to update and optimize the models. Because the labels of training images are given, the deep learning-based image segmentation methods are mainly focused on semantic segmentation, which not only performs segmentation but also performs classification on each pixel. Among all deep-learning image classification methods, the convolutional neural network (CNN) is widely used. Thus, deep learning-based segmentation models are mostly originated from CNN and its modification form, the fully convolutional network [24] (FCN).

With the growth of deep neural network, the FCN is a general approach to semantic segmentation. Recently, the graph convolution network [25] (GCN) was proposed to perform convolution operations on the graph-structured-data (each node is a superpixel) instead of on the pixels directly to improve the efficiency. Briefly speaking, suppose that the number of nodes in a graph is N, a single graph convolutional layer can be written as:



(3.1)

where *H*(*l*) denotes feature maps in the *lt*h layer of the network and *W*(*l*) is a learnable weight matrix of *N* nodes,  where *A* and *IN* are the similarity matrix of the graph and the identity matrix.  is the degree matrix derived from . is the nonlinear activation function (e.g., ReLU) enabling the nonlinearity the of the GCN.  captures neighbors of nodes and  gives parameters for multiplications, and the combination of these two operations is like standard convolutional operations in the CNN.

After the work in [25], several semantic segmentation methods apply the concept of the GCN by transforming original feature maps to graphs (graph representations) and viewing the data as graphs. Then, the graph convolution is performed on the graph space and transformed back to 2D feature maps, which can connect to subsequent standard convolutional layers. Integrated with FCN architecture, deep learning models attached with layers using the concept of the GCN to improve the performance of semantic segmentation. In Section 3, we will review some classic CNN-based methods for semantic segmentation, and discuss several methods using the GCN to perform semantic segmentation.

Table 2. Summary of methods using GCN in Section 3.

|  |  |  |
| --- | --- | --- |
| Methods | Procedure | Advantages |
| GCU [31] | Project and reproject features by defined formula | (1) Projection and reprojection are explainable  (2) Less learned parameters |
| GloRe [36] | Project and reproject features by gradient descent | Practically simple |
| SpyGR [33] | (1) Generate adjacency matrix by gradient descent  (2) Spatial pyramid for multiscale pattern | (1) Without projection and reprojection steps  (2) Capture multiscale information |

**3.1 Classic CNN-Based Segmentation Methods**

**3.1.1 Fully Convolutional Networks for Semantic Segmentation (FCN)**

The FCN [24] is the first deep learning-based semantic segmentation model, it replaces fully connected layers at the final part of the CNN models such as the AlexNet, the VGG, and the GoogleNet with convolutional layers to perform classification (prediction) at every pixel. The FCN up-samples the final prediction result by several times and uses the skip architecture to refine the semantic segmentation results. The upsampling operation of the FCN consists of the filterd initialized with the bilinear kernel. Fig. 3.1 shows the rough structure of FCN.

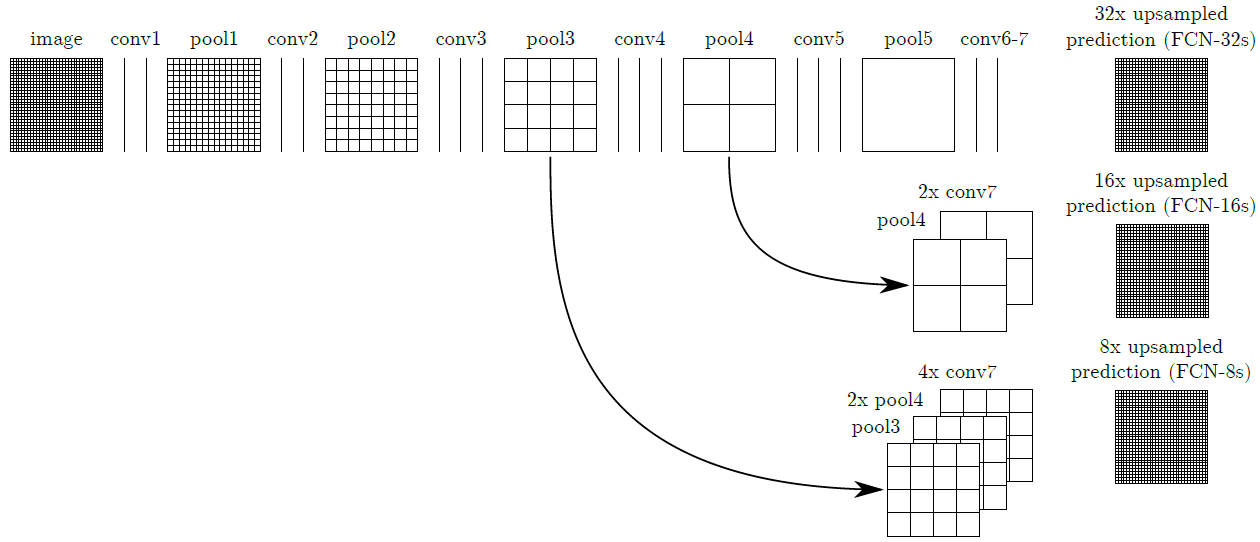


Fig. 3.1. The FCN model structure [24].

**3.1.2 Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation (DeepLabV3+)**

On the object boundaries or details, the results of the FCN may not achieve precise predictions after upsampling because the resolution of the feature map before upsampling is too low (i.e. the size of the feature map is too small). Chen *et al.* [26, 27] proposed a decoder structure to gradually recover the details of segmentation results. In DeepLabV3+ [26], atrous convolution mentioned in [27] is applied to obtain a larger receptive field while retaining the resolution of feature maps. Moreover, it applies depth-wise separable convolution proposed by [28] to reduce the computation costs of convolution operations. Fig. 3.2 shows the rough structure of DeepLabV3+. It can predict the labels more precisely on the object boundaries compared to the FCN.

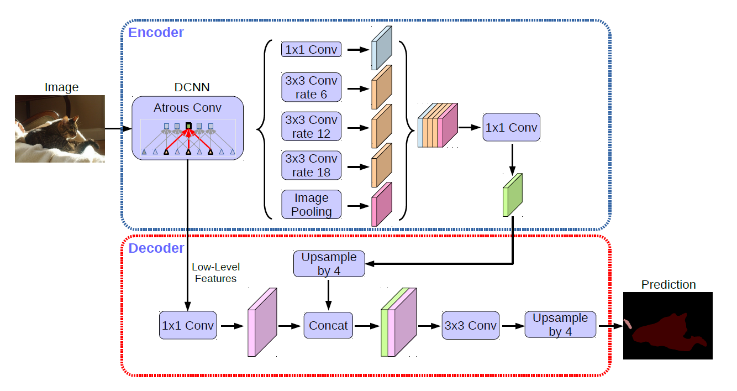


Fig. 3.2. The structure of the DeepLabV3+ model [26]. The encoder part applies atrous convolution to the standard DCNN (e.g., the ResNet), and uses parallel convolution layers with different atrous rates to capture multi-scale context of the image. The decoder part is a simple structure, which is a concatenation of feature maps and convolutions.

**3.2 Beyond Grids: Learning Graph Representations for Visual Recognition (GCU)**

Inspired from region-based image recognition, Li *et al.* proposed a framework, called the graph processing unit [31] (GCU), that projects an original 2D feature maps to a graph (graph representation, graph domain, graph space), performs graph convolutions on the graph, and transforms back to 2D feature maps. In the graph, the nodes are the clusters of pixels (i.e., superpixels) and the edges are the similarities between superpixels. Li *et al.* [31] stated that convolutions in the graph domain can better captures the long-range information of the original image. Fig. 3.4 shows the structure of a GCU, which mainly consists of three steps: (1) Graph projection, (2) Graph convolution, and (3) Graph reprojection. The procedure of the GCU can be formulated as:



(3.2)

where G with subscripts denotes each of the steps and and are the inputs and outputs of the GCU, respectively. Inputs of GCU are feature maps from the deep CNN, which can regard as matrices, and the size of outputs is equal to that of inputs, which can be easily integrated to existing deep learning segmentation models. Each step will be introduced in the next paragraph.

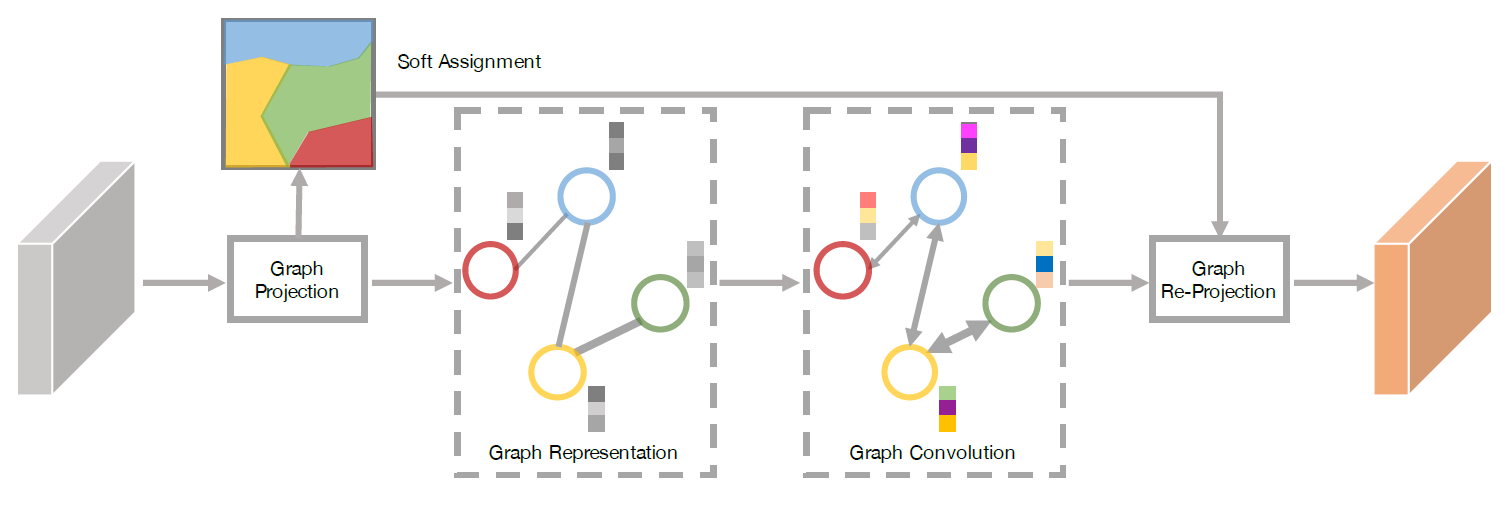


Fig. 3.4. An illustration of the GCU structure [31].

**Graph Projection**

Assume that the input feature map X has the size of H×W, and each entry of X, denoted by *xij*, which represents a feature vector at pixel (*i*, *j*) with dimension *d*.

In the process of *Gproj*, X is first assigned to a set of vertices by computing a soft-assignment coefficient of a feature vector *xij* to the center of the *k*th cluster wk :



(3.3)

where || ||2 denotes the *L*2-norm and *σk* can be regarded as the standard deviation of the cluster *k*. It computes the weighted Euclidean distance between xij and *wk*, indicating that if a pixel (*i*, *j*) with a normalized feature vector should belong to the *kt*h cluster, will be relatively large, so the soft assignment is somewhat like the probability of pixel (*i*, *j*) belonging to the *k*th cluster, and it assigns pixels with similar features to the same vertex.



(3.4)

Based on , we compute , which can be viewed as the weighted sums of normalized feature vectors by means and standard deviations, with weights emphasizing the importance of pixel features similar to cluster centers. is further *L*2-normalized to , which represents the feature vector of k-th node. forms *kt*h columns of the feature matrix Z, with size d×|V| (|V| is the number of nodes in the graph). After that, compute the adjacency matrix A of the graph:



(3.5)

**Graph Convolution**

After getting feature vectors of nodes of the graph Z and corresponding adjacency matrix A, perform graph convolution as introduced at the beginning of Section 3:

****

(3.6)

where *A* is the adjacency matrix computed from Eq. (3.5), *ZT* is the feature matrix consisting of each node in the graph, *Wg* is a weight matrix with parameters learned by training data, and f is the nonlinear activation function.

**Graph Reprojection**

In this step, we reverse the graph projection step and transform the vertex features in the graph back to the original 2D space. Compute pixel features of feature maps by weighted sums of vertex features in the graph:

****

(3.7)

where  is the matrix composed of which defined in Eq. (3.3). The size of the output is *HWd*. After rearranging, it matches the size of input X.

The above three steps compose a GCU. Li *et al.* [31] proposed to attach the GCU to the FCN using a ResNet [32] backbone to improve the segmentation results.

**Results**

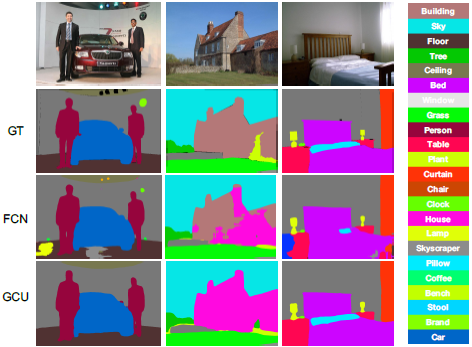


Fig. 3.5. Semantic segmentation results of the FCN and the GCU [31]. The second row is the ground truth, the third is results of FCN, and the fourth is results of FCN using ResNet50 attached with the GCU. We see that the GCU generate a smoother prediction compared to the FCN.

**3.3 Graph-Based Global Reasoning (GloRe) Networks**

Similar to the GCU described in Section 3.2, the graph-based global reasoning (GloRe) [36] network adopts feature transformations and uses the GCN [25] to better capture the global relations (i.e. long-range dependencies) for semantic segmentation. Feature maps in the coordinate space are transformed to the interaction space (graph), performed graph convolutions, and transformed back into the coordinate space. Different from the GCU [31], Chen *et al.* [36] proposed a novel structure, GloRe, which can also output feature maps with the same size as that of the input feature map. Fig. 3.6 shows the structure of GloRe and the details of the process are introduced in the next paragraph.

****

Fig. 3.6. Structure of GloRe unit [36]. The middle part with background colors indicates the graph convolutions.

**Graph Projection**

Let X be the input feature map with dimension *LC*, where *L* equals to *H* (height)\**W* (width), and *C* is the dimension of each pixel feature. In short, the transformed feature is computed as follows:

(3.8)

where *vi* is the *i*th transformed feature and *bij* is a learning weight of the transformed opration.

**Graph Convolution**

Chen *et al.* [36] proposed to treat transformed features V as nodes in a fully connected graph, and learn edge weights of the graph. In particular, the adjacency matrix Ag of the graph and filter weights *Wg* of the GCN are learned simultaneously by the gradient descent during training. GloRe uses G = I-Ag as the adjacency matrix to prevent from optimization difficulties (e.g. gradient vanishing). The GCN layer in GloRe is as follows:

.

(3.9)

**Reverse Projection**

Similar to the GCU, GloRe reuses the B matrix generated by the projection function, and projects the features in the graph back to the coordinate space:

(3.10)

where Y is the output of the reverse projection.

**Results**

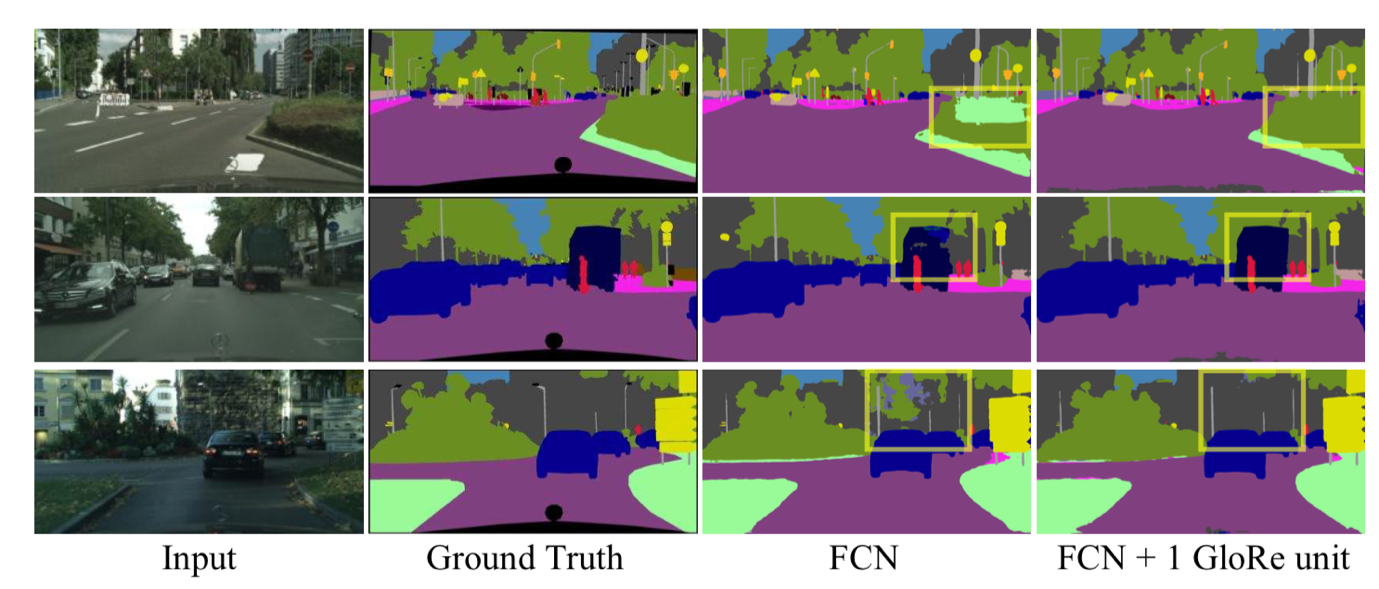


Fig. 3.7. Visual comparisons of FCN and FCN attached with GloRe [36]. We see that GloRe can better capture global relation and generate smoother semantic segmentation results.

**3.4 Spatial Pyramid-Based Graph Reasoning for Semantic Segmentation (SpyGR)**

Different from previously mentioned methods, Li *et al*. [33] proposed the spatial pyramid-based graph reasoning (SpyGR) layer, which directly performs graph convolutions on original feature maps without the steps of projection and reprojection. It also adopts the spatial pyramid structure, which performs graph convolutions at different scales, to capture the multiscale long-range dependencies of the patterns in an image (described in Section 2.5). Fig. 3.8 illustrates the framework of the SpyGR layer.

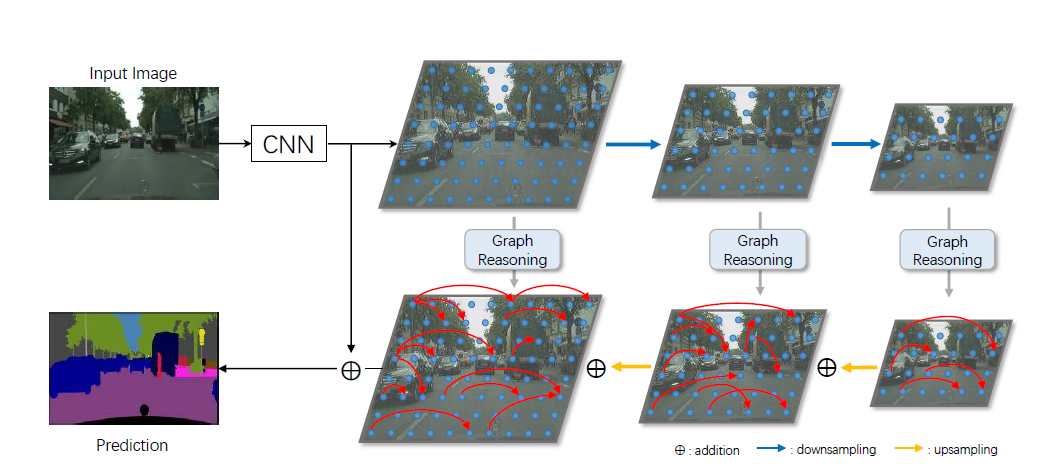


Fig. 3.8. A diagram of the SpyGR layer [33]. Inputs are feature maps from deep CNN, and outputs the prediction (segmentation) map of the image. The red arrows indicate long-range dependencies between different patterns marked by blue dots.

Let X be the input feature matrix, with size (H\*W)\*C (similar to that in Section 3.2), and be a linear embedding of X realized by CNN convolution layers to reduce the dimension of each entry of X from C to M and then is followed by ReLU nonlinearity. Calculate similarities using inner products between position *i* and *j* under a lower dimension *M* by:



(3.11)

where (X) is a diagonal matrix with diagonal entries as attention coefficients for inner products, and subscripts indicate positions. This step also resembles the projection and the reprojection step used in GCU [31] and Glore [36]. However, the adjacency matrix is of the size (H\*W)\*(H\*W), which matches the original image size, and original features instead of projected features are used to perform graph convolutions in Eq. (3.13). This is why the author said that SpyGR requires no projection and reprojection steps.

After obtaining the adjacency matrix , an improved Laplacian matrix is applied to perform graph convolution:



(3.12)

where is a diagonal matrix with i-th elements . is data dependent because different images produce different and . Only the dimension reduction operations and are learned and then fixed during inference.

Finally, conduct the graph convolution operation:



(3.13)

where X and Y are the inputs and the outputs, is the nonlinear function (e.g. ReLU), and is a learnable weight matrix as described at the beginning of Section 3.

**3.5 Other Graph-based Segmentation Methods Using Deep Learning**

There are many other deep learning-based semantic segmentation methods using the GCN [25] by designing different modules to better capture global contexts of images. They design different methods for graph construction, graph convolution, and reverse construction.

The dual graph convolution network (DGCNet) [37] applies a two-branch GCN approach, which transforms input feature maps into two different graphs. One models spatial relationships by down-sampling feature maps along spatial dimensions, while the other one models channel dependencies by learning projection functions and projects input features into a graph with lower feature dimensions. After reversely projecting these two-branch features, the DGCNet merges features by summing two-branch features and original input features, producing output features with the size identical to that of the input.

The symbolic graph reasoning (SGR) [38] method conducts graph construction of input features in a similar way to that of the GCU [31], but uses additional off-the-shelf context relationship graphs to help performing graph convolution. It embeds an object to a linguistic vector, and attaches to each node of the graph by concatenating transformed features of inputs along feature dimension. After reversely projecting graph features to the input space, the SGR adds the feature and original inputs to form outputs of the SGR whose size is the same as that of the input for convenience of integration to existing semantic segmentation networks.

**4. Conclusion and Further Reading**

In this tutorial, we introduced the basic concepts of segmentation by the graph theory. For traditional segmentation methods, they seek to solve the optimization problem to find the optimal clustering (optimal segmentation results) of an image. For deep learning semantic segmentation methods, they learn parameters of models using the training data and further apply the GCN to their proposed models. Although these traditional and modern deep learning methods solve different segmentation problem, they both used the concept of the graph theory to improve the performance.

Each subsection of this tutorial is a paper I used to survey. If one wants to know more about conventional methods and modern methods discussed in the tutorial, please refer to my reference in each subsection for more explanations of each method. In fact, the GCN is a branch of the graph neural network (GNN) [42], so if one wants to know more about the graph neural network and its variations, please refer to [39] for a brief but complete introduction, with [40] and [41] for better comprehension. If one wants to know more about basic graph signal processing and the spectral graph theory, please refer to [4] for more details.

**References**

[1] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Trans. PAMI,* 22(8):888–905, 2000.

[2] G.H. Golub and C.F. Van Loan, *Matrix Computations.* John Hopkins Press, 1989.

[3] <https://www.itread01.com/content/1542078571.html>

[4] Stanković, Ljubiša, Miloš Daković, and Ervin Sejdić. Introduction to graph signal processing. *Vertex-Frequency Analysis of Graph Signals.* Springer, Cham, 2019. 3-108.

[5] P. Felzenszwalb and D. Huttenlocher. Efficient graph-based image segmentation. *IJCV*, 59(2):167–181, 2004.

[6] <https://en.wikipedia.org/wiki/Pixel_connectivity>

[7] Z. Li, X.-M. Wu, and S.-F. Chang, Segmentation using superpixels: A bipartite graph partitioning approach. *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2012, pp. 789–796.

[8] I. Dhillon. Co-clustering documents and words using bipartite spectral graph partitioning. *KDD*, 2001.

[9] H. Zha, X. He, C. Ding, H. Simon, and M. Gu. Bipartite graph partitioning and data clustering. *CIKM*, 2001.

[10] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Trans. PAMI*, pages 603–619, 2002.

[11] <https://towardsdatascience.com/spectral-clustering-aba2640c0d5b>

[12] M. Liu, O. Tuzel, S. Ramalingam, and R. Chellappa. Entropy rate superpixel segmentation. *Proc. of CVPR*, pages 2097– 2104, 2011.

[13] X. Wang, Y. Tang, S. Masnou and L. Chen, A Global/Local Affinity Graph for Image Segmentation. *IEEE Transactions on Image Processing*, vol. 24, no. 4, pp. 1399-1411, April 2015.

[14] Y. C. Pati, R. Rezaiifar and P. S. Krishnaprasad. Orthogonal matching pursuit: recursive function approximation with applications to wavelet decomposition. *Proceedings of 27th Asilomar Conference on Signals, Systems and Computers*, 1993, pp. 40-44.

[15] X. Zhang, Z. Wei, J. Feng, and L. Jiao. Sparse representation-based spectral clustering for SAR image segmentation. *Proc. SPIE*, vol. 8006, Nov. 2011, Art. ID 800608.

[16] Zhengqin Li and Jiansheng Chen. Superpixel segmentation using Linear Spectral Clustering. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015

[17] R. Achantan, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Susstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE Trans. on PAMI*, 34(11):2274– 2281, 2012.

[18] Gao, L., J. Song, F. Nie, F. Zou, N. Sebe, and H. T. Shen. Graph-Without-Cut: An Ideal Graph Learning for Image Segmentation. *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, Feb. 2016.

[19] Fan, K. 1949. On a Theorem of Weyl Concerning Eigenvalues of Linear Transformations. I. *Proceedings of the National Academy of Science* 35:652–655.

[20] M. B. Salah, A. Mitiche and I. B. Ayed. Multiregion Image Segmentation by Parametric Kernel Graph Cuts. *IEEE Transactions on Image Processing*, vol. 20, no. 2, pp. 545-557, Feb. 2011.

[21] I. B. Ayed, A. Mitiche, and Z. Belhadj. Polarimetric image segmentation via maximum-likelihood approximation and efficient multiphase level-sets. *IEEE Trans. Pattern Anal. Mach. Intell*, vol. 28, no. 9, pp. 1493–1500, Sep. 2006.

[22] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Trans. Pattern Anal. Mach. Intell*, vol. 23, no. 11, pp. 1222–1239, Nov. 2001.

[23] X. Wang, H. Li, C. Bichot, S. Masnou, and L. Chen. A graph-cut approach to image segmentation using an affinity graph based on l0- sparse representation of features. *IEEE International Conference on Image Processing*, 2013, pp. 4019–4023.

[24] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015.

[25] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *ICLR*, 2017.

[26] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. *ECCV*, 2018.

[27] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *TPAMI*, 2017.

[28] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv:1704.04861, 2017.

[29] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *The J. Mach. Learn. Res*., vol. 6, pp. 1453–1484, 2005.

[30] F. Liu, G. Lin, and C. Shen. CRF learning with CNN features for image segmentation. *CoRR*, abs/1503.08263, 2015.

[31] Y. Li and A. Gupta. Beyond grids: Learning graph representations for visual recognition. In *NeurIPS*, pages 9245–9255, 2018.

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

[33] Li, X., Yang, Y., Zhao, Q., Shen, T., Lin, Z., Liu, H.: Spatial pyramid based graph reasoning for semantic segmentation. In *CVPR*, 2020.

[34] Maoke Yang, Kun Yu, Chi Zhang, Zhiwei Li, and Kuiyuan Yang. Denseaspp for semantic segmentation in street scenes. In *CVPR*, pages 3684–3692, 2018.

[35] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *CVPR*, pages 2881–2890, 2017.

[36] Yunpeng Chen, Marcus Rohrbach, Zhicheng Yan, Yan Shuicheng, Jiashi Feng, and Yannis Kalantidis. Graph-based global reasoning networks. In *CVPR*, 2019.

[37] Zhang, L., Li, X., Arnab, A., Yang, K., Tong, Y., Torr, P.H. Dual graph convolutional network for semantic segmentation. In *BMVC,* 2019.

[38] Liang, X., Hu, Z., Zhang, H., Lin, L., Xing, E.P. Symbolic graph reasoning meets convolutions. In *NeurIPS,* 2018.

[39] <http://speech.ee.ntu.edu.tw/~tlkagk/courses/ML2020/GNN.pdf>

[40] <https://www.youtube.com/watch?v=eybCCtNKwzA>

[41] <https://www.youtube.com/watch?v=M9ht8vsVEw8>

[42] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 2009.