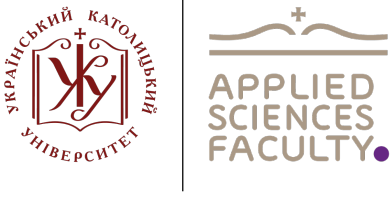
Ukrainian Catholic University
Faculty of Applied Sciences
Computer Science Undergraduate Program

Spectral Partitioning of the Image

Linear Algebra Final Course Project

Authors:
Butynets Danylo,
Semchyshyn Pavlo,
Sultanov Andriy



Spectral Partitioning of the Image

Butynets Danylo, Pavlo Semchyshyn, Sultanov Andriy

28 April 2021

Abstract

The spectral approach to clustering will be discussed in this report alongside with how it can be used in large-scaled applications such as image segmentation. The code and results can be found here: https://github.com/psemchyshyn/SpectralClusteringL

1 Introduction

Image segmentation became one of the most popular preprocessing task over the last decades. Performing segmentation requires a reliable clustering algorithm in order to effectively partion the image. Hundreds of clustering techniques arose in the last 20 years. Generally, we can relate each of them to the next categories: partitional clustering, hierarchical clustering and density-based clustering. For image segmentation, the famous K-means algorithm (which belongs to partitional clustering category) remains probably the most popular as its time complexity is close to linear and it is extremely suitable for mining large-scaled application (image partitioning, for instance). On the other hand, it is only applicable to the data, which is convex and performs very poorly on non-linear shaped data. Spectral clustering, which is another partitional clustering algorithm, doesn't have this drawback. In this report, we try to embed this approach into our task with the goal to get an improved segmentation of the complex images (comparing to that, which K-means suggests).

2 Problem formulation

Suppose we are given an image with a few distinct objects on it. Object represented in the image is basically just a set of similar pixels, so that with each distinct object there will be a one set of pixels associated. Obviously, these sets of pixels corresponding to different objects should not intersect. Such sets of pixels are called clusters and identifying clusters is our primary goal, as they give us distinct objects represented in the image. Now suppose we have some metric to define whether two pixels are similar or not. Given that, we can represent the image as an undirected graph, whose nodes are all the pixels of the image and there is an edge between two nodes if the corresponding pixels are similar (It is possible and actually it is implemented in such way that the edges have

Such metric will be described in the implementation pipeline section

weight depending on how similar two pixels are). With introducing graph as the representation of the image, we can formulate our task in a strict mathematical way as following. Given an undirectional graph $G = (V, E)$, we need to find such subsets of vertices $\{A_1, A_2, A_3, ..., A_n\}$, where $A_i \subset V, 1 \leq i \leq n$, $\bigcup\limits_{1 \leq i \leq n} A_i = V$ and $A_i \cap A_j = \emptyset$ such that the amount of edges between sets remain as small as possible and the amount of edges inside sets are as large as possible. From the first point of view, the final goal can resemble the task of finding minimal cut in the graph, where cut is defined as $cut(A_1, A_2) = \sum_{v_i \in A_1, v_j \in A_2} W_{ij}$, where $A_1 and A_2$ are non-intersecting subsets of V(and their union is equal to V) and W_{ij} is the value of edge between these vertices (For more than two subsets the definition of cut can be the following $cut(A_1, A_2, ..., A_k) = \sum_{i=1}^{k} cut(A_i, \overline{A_i})$, where $\overline{A_i}$ stands for a set $V \setminus A_i$). Such problem is known as max-flow min-cut task and can be easily solved with Ford-Fulkerson algorithm for example. (**?**)

However, this approach to clustering would fail, as the target function doesn't put any restrictions on the size of the clusters and would likely choose some isolated vertices and not meaningful partitions. We could modify the target function taking into account the size of the clusters. The target function could look like $f(A_1, A_2, ..., A_3) = \sum_{i_1}^{k} \frac{cut(A_i, \overline{A_i})}{|A_i|}$, ($A_i$ is the power of a set) which is called RatioCut. Finding a minimum of this function is an NP-hard problem. Spectral clustering is a convenient heuristics, which gives us a very good approximation to the Ratio Cut optimization problem.

Note: There is also another choice of a target function called Normalized Cut. Though, it requires modifications to the basic ideas of our approach and we decided not to include it in the report

3 Algorithm of spectral clustering

3.1 Justification of the algorithm choice

The reason, why we decide to choose spectral clustering over traditional clustering approaches such as k-means is that it is able to deal with data of much more complex form, for instance, spiral datasets or any other nonlinear shapes, because spectral clustering doesn't make assumptions about the shape of the data.

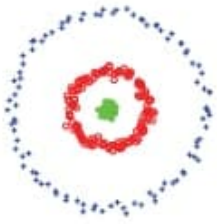


Figure 1: Spiral shape data example

As the example, logical clusters in the above picture (denoted by different

colors) can't be obtained by k-means, however it is not a problem for spectral clustering.

3.2 Setup

As the name suggests, spectral clustering works with the spectrum of some matrix. This matrix is called Laplacian matrix. We start from a graph G = (V, E). The adjacency matrix of this graph is denoted by W. It contains the values of edges between vertices. We also need a degree matrix D, which is a diagonal matrix, with entries equal to $d_i = \sum_j^k W_{ij}$ - degree of vertix(sum of of the weights of edges incident to the vertex). Then, matrix L = D - W is called unnormalized Laplacian. That is exactly the matrix, in spectrum of which we are interested (Note! unnormalized Laplacian is used for solving ratio cut problem, but there is also normalized Laplacian, which is used to solve also mentioned normalized cut problem) There are many interesting properties of matrix L. First of all, it is symmetric. Then, there is a unique quadtratic form associated with this matrix. Understanding of how we present this quadtratic form is a key to understanding the algorithm.

$x^T Lx = \sum_{i,j=1}^n L_{ij} x_i x_j = \sum_{i,j=1}^n (D_{ij} - W_{ij}) x_i x_j = \frac{1}{2}(\sum_{i=1}^n D_{i,i} x_i^2 + \sum_{i=1}^n D_{i,i} x_i^2 - \sum_{i,j=1} W_{i,j} x_i x_j) = \frac{1}{2} \sum_{i,j=1}^n (x_i - x_j)^2$

As we see, the quadratic form is always greater or equal to zero, which means that L is positive semidefinite. Therefore we can list the eigenvalues as $0 = \lambda_1 <= \lambda_2 <= ... <= \lambda_n$

3.3 Connection to the RatioCut problem

Let's consider an example, when we already have a partition into K optimal clusters $A_1, A_2, ...A_k$. We can denote it by using set of K vectors $\{x_i\}_i^k$, where each component of i_{th} vector is $x_{ij} = \frac{1}{\sqrt{|A_i|}}$ if $v_j \in A_i$ else 0. Then, the value of quadratic form in x_i is:

$x_i^T Lx_i = \frac{1}{2} \sum_{l,j=1}^n (x_l - x_j)^2 = \sum_{v_j \in A_i, v_l \notin A_i} W_{j,l} (\frac{1}{\sqrt{|A_i|}} - 0)^2 + \sum_{v_j \in A_i, v_l \notin A_i} W_{j,l} (0 - \frac{1}{\sqrt{|A_i|}})^2 = \frac{1}{2}(\frac{cut(A_i, \overline{A_i})}{|A_i|}) + \frac{1}{2}(\frac{cut(A_i, \overline{A_i})}{|A_i|}) = \frac{cut(A_i, \overline{A_i})}{|A_i|}$

Then,

$\sum_{i=1}^k x_i^T Lx_i = \sum_{i=1}^k \frac{cut(A_i, \overline{A_i})}{|A_i|} = RatioCut(A_1, A_2, ...A_k)$

Considering that the initial clusters were optimal, we obtain the minimal Rati-

oCut from the above vectors $\{x_i\}_i^k$. A very important observation is that this set is orthonomal. Next, because we allowed x_{ij} to take only discrete values, the problem to find them turns out to be NP-hard as well. However, if we let x_{ij} take arbitrary real values, we can sum everything up as the following: in order to minimize the ratio cut, each component of the sum $\sum_{i=1}^{k} x_i^T L x_i$ ($\frac{cut(A_i, \overline{A_i})}{|A_i|}$) should be minimized. Moreover, we found out that $\frac{cut(A_i, \overline{A_i})}{|A_i|} = x^T L x$ (with x_i of unit length and each next x_j being orthogonal to the previous x_is). What we have is:

$\frac{cut(A_1, \overline{A_1})}{|A_1|} = \min_{||x||=1} x^T L x = \lambda_1$ - the first eigenvalue of the Laplacian matrix. Denote by x_1 - the argument, which minimizes above equation. That is the corresponding eigenvector.

$\frac{cut(A_2, \overline{A_2})}{|A_2|} = \min_{||x||=1, x \perp x_1} x^T L x = \lambda_2$ - the second eigenvalue of the Laplacian matrix. Analogically, denote by x_2 the corresponding eigenvector.

...

$\frac{cut(A_k, \overline{A_k})}{|A_k|} = \min_{||x||=1, x \perp S_{k-1}} x^T L x = \lambda_k$ - the K eigenvalue of the Laplacian matrix. S_k is a linear span of the first K - 1 eigenvectors.

Then, minimized value of RatioCut function can be then approximated by $\sum_{i=1}^{k} \lambda_i$, and the clusters can be determined with first K eigenvectors.

3.4 How we classify

For now, we know that for clustering into K segments, K smallest eigenvalues are used. However, it may still be unclear how exactly K eigenvectors are used for labeling into clusters. (As we, so to say, lost the opportunity for straightforward labeling by allowing them to take any real values). There are basically two approaches to deal with this problem.

The first one is called recursive bipartitioning. The idea is to start with two clusters, then only one eigenvector will be used for clustering. By choosing some threshold, for instance, the median (then we agree on clusters having equal size) of this eigenvector components, we can classify our nodes in the following way: if i_{th} component of the vector is less than the threshold, then i_{th} node goes to the first cluster. If it is bigger than the threshold, then it goes to the

second cluster. If we wanted more than 2 clusters, we can go deeper and perform same actions now on these two newly formed clusters - that is why it is called recursive bipartitioning. As we see, this approach doesn't even require top K eigenvectors, however it is much less effective.

The other approach implies using k-means. Having k first eigenvectors, we can represent i_{th} node of the graph as the vector constisting of i_{th} components of these k eigenvectors. Then, running k-means on these new representation of the data points (nodes of the graph) gives us wanted clusters. As it turned out, such approach is known as laplacian eigenmap - quite popular non-linear dimensionality reduction technique. This approach is used in our project.

3.5 Drawbacks of spectral clustering

The drawback of this algorithm is a time complexity and memory cost that it requires. Building an adjacency matrix is $O(n^2m)$ and eigenvalue decomposition is $O(n^3)$ for time complexity (n is the amount of data points, m - their dimension). This is unacceptable for large-scale applications(such as image segmentation). Many problems arose trying to come up with a solution to this. This will be highlighted in the next section.

4 Implementation Pipeline

4.1 Tools

In order to create a working prototype of the program, we decided to write it using Python programming language. In the Github link inserted at the beginning, one can find the reference to the Google Colab document with the all code used for this task.

4.2 Image preprocessing

We start by reading the image as the matrix each element of which is represented by its RGB values. As denoted in the last section, we have a problem of processing image, both time and space complexity suffers. Fortunately, there are many ways to handle this. We decided to stop by the next one: we can work not with all pixels of the image, but with a fewer by merging similar pixels into one bigger called "superpixel" with the help of some quick algorithm such a k-means. The only problem with this approach is that the number of superpixels is entered manually by us, and not determined by any algorithm (we observed that for different images, different amount of superpixels are optimal). Below are some examples of partitioning into superpixels, as well as plotting of superpixels onto 3D space based on their RGB values.

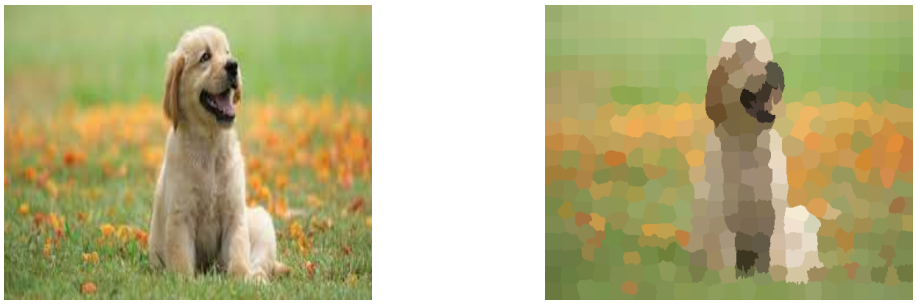


Figure 2: Example of image after making superpixels. Number of superpixels = 400

Figure 3: Example of image after making superpixels. Number of superpixels = 3000

4.3 Graph representation of the image

There are at least three possible ways to construct a graph from the image and there aren't any theoretical studies focusing on choosing what way is the best in general. Those approaches are: K-nearest neighbor graph, ϵ-neighborhood graph and fully connected graph. We won't stop on the last two, as by experimenting, KNN worked best for us. (The general idea is that v_i is connected with v_j when v_j is among the K-nearest neighbors of v_i, or vi is among the K-nearest neighbors of v. The distance between the data points is based on the Euclidian distance between the pixels. Taking this graph will generally work for our task. However, the Euclidian distance is not the best similarity measure between the data points. For this reason, for the weights of the edges of the graph we chose Gaussian kernel. We won't discuss its properties here, as it is a little bit beyond the scope of our project. For what one should think about Gaussian kernel is that it is like normalization function for Euclidean distance, which gives a good similarity measure definition. The graph built by KNN approach will be used to construct the Laplacian matrix. Also, not hard not notice that we need to choose k - the numer of neighbours. Unfortunately, there is no algorithm to define this k. We tried different values, k = 30 worked best for us.

Figure 4: Superpixels plotted onto 3D space (400 and 3000 segments respectively for dog image and Korean guy)

4.4 Laplacian matrix

Given a graph, it is trivially easy to get its Laplacian matrix. As denoted in the theoretical part, it is just the difference between adjacency matrix and degree matrix of our graph.

4.5 Number of Clusters

The theory suggests, as explained before, that for clustering into k groups we require k first eigenvalues. However, it is not always the best choice in practice. And, if we want for the program to define the best possible number of clusters, then, how it can be done? By default, when we expect for user to manually choose the number of clusters, we follow the theoretical approach. On the other hand, the implementation of the automatic number of clusters choice lacks precision. The approach here is the following. The best practices say here that, in general, the most meaningful eigenvalues are those that happen before the biggest eigengap(the distance between two consequent eigenvalues). Therefore, if $\lambda_n - \lambda_{n-1}$ is the biggest, then we select all n top(smallest) eigenvalues. Now, remember that we use k-means on the new representations of the points. There are many different metrics that define how good k-means is for a particular amount of clusters. Again, discussing them is beyond the scope of the project. We tried several such metrics and stayed by silhouette coefficient. However, we haven't obtained a good result here. The eigengap approach wasn't really efficient in our task. The gap is usually on index more than 300, it will lead to ridiculously large eigenmap and k-means performs poorly this way. On the other hand, choosing manually the number of eigenvalues (experimenting with different values) makes silhouette coefficient work better and gives reasonable results.

4.6 Propagating results on superpixels to the real ones

Remember that all that we have done so far was for the superpixels, not the real ones. The propagation of the results to real pixels is quite straightforward.

We have a cluster of superpixels, then we collect all the real pixels belonging to them into one set - this will be a cluster on the initial image.

4.7 Problem of Manual Input

What we consider as the biggest problem of this program is that the final result and quality of clustering are heavily dependent on values the user inputs (especially number of superpixels and clusters as well as the number of neighbours for KNN), and sometimes it requires tuning of these to acquire a desirable result. Below you can witness some example of clustering in image, graph and superpixel RGB forms.

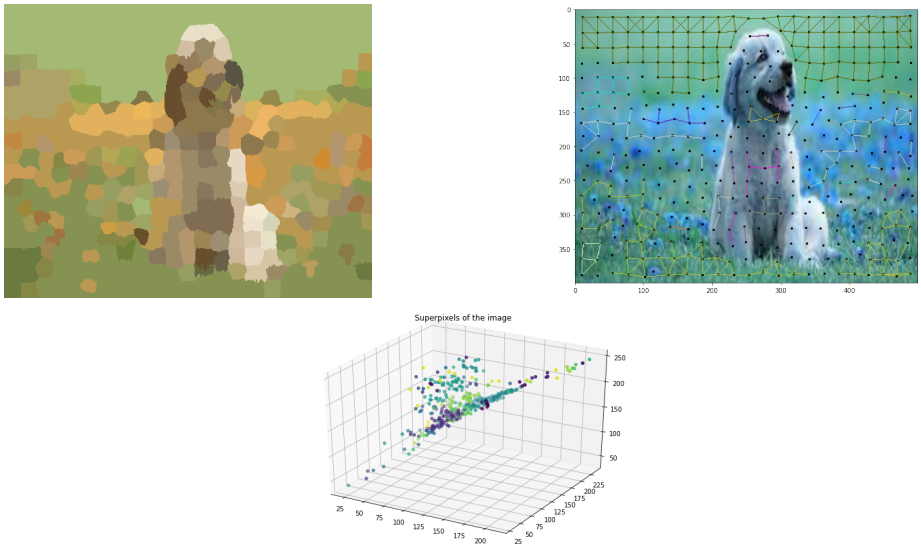


Figure 5: 1. Clustered image (40segments); 2. Original image with partitioned graph overlay; 3. Superpixel partitioning in 3D space after clustering. (Manual input of user, K = 50)

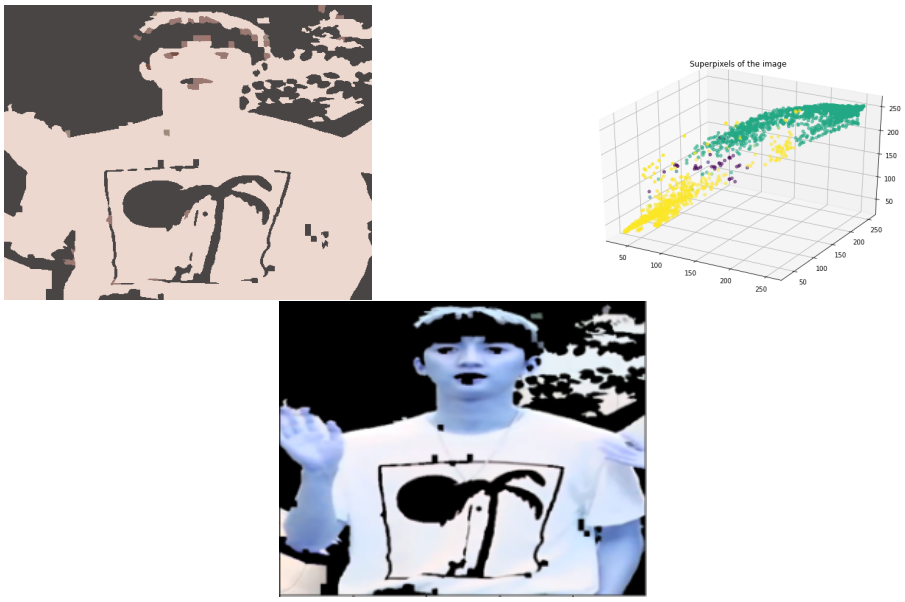


Figure 6: 1. Clustered image (3 clusters); 2. Superpixels after clustering in 3D space; 3. Biggest cluster on the original image, other clusters replaced with black. (Numbers of clusters is chosen automatically)