An Exploration of Image Segmentation via Spectral Clustering

Jessica Qiu and Angela Wu

December 2022

Contents

1 Introduction

1.1 Spectral Clustering

Spectral clustering is one of the most widely used data analysis tools. This technique divides the data into groups, or clusters, according to their similarity, so data in the same group are considered similar while data in different groups are not similar to each other [1]. Now the question is, how do we determine these clusters? Spectral clustering represents the data sets as graphs and classifies them by the edges connecting the points, so the points that are most connected to each other are considered to be part of the same group. Thus, while other types of clustering like the k-means algorithm rely on Euclidean distance, spectral clustering is more graph-based and depends on the connections within data points in its graph [5].

1.2 Image Segmentation

For our project, we specifically chose to explore image segmentation via spectral clustering. While the underlying theory is the same, image segmentation focuses on partitioning the image into regions, which ideally represent distinct objects, by detecting groups of pixels within an image by considering how similar they are with regards to properties like intensity, color, and position. Image segmentation has many useful applications including medical imaging, face recognition, video surveillance, and many more [6].

2 Constructing the Similarity Matrix

2.1 The Data Set

Our data set was an image of two circles in grayscale. The goal was to apply image segmentation and be able to detect and differentiate the shapes within the image. However, while our code used an image as an input, it can be generalized to take in various types of data sets.

To start, we resized the image by a factor of $\frac{1}{10}$ so that it could be easier to work with, as an image with too many pixels would take too long to run.

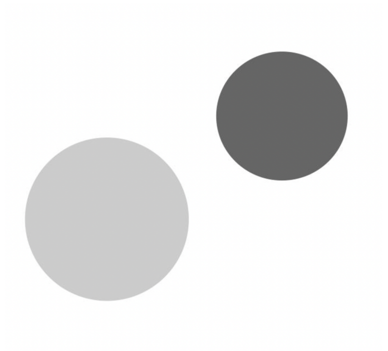


Figure 1: Original image

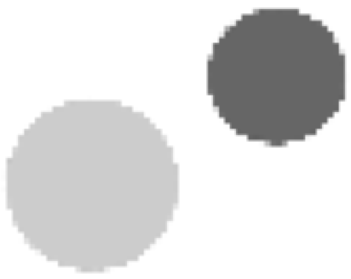


Figure 2: Resized image

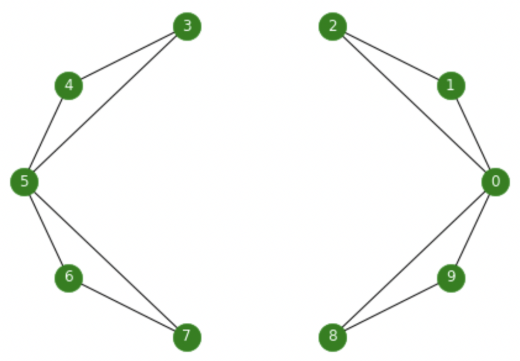
After getting the new, more pixelated image on the right, we created a $3 \times n$ matrix (n denotes the total number of pixels in the image) where the columns of the matrix contained the row, column, and intensity of each pixel since we wanted to use these three factors to determine how similar two pixels are for our k-nearest neighbors algorithm. This is shown in the code below:

```
m = zeros(3, s*s)
for i in 1:s
    for j in 1:s
        m[:,s*(i-1)+j] = [i, j, red(new_img[i, j])]
    end
end
```

2.2 Understanding Similarity Graphs and Similarity Matrices

To understand why we coded the way we did for the next step, it's important to start with the basic reasoning for obtaining a similarity matrix from a similarity graph. A similarity graph shows how all the nodes of a data set are related to each other. The image below is an example of a similarity graph with 8 nodes and two disconnected components taken from an article on spectral clustering [1].

This example and our implementation use unweighted similarity matrices, matrices which only consist of 1s and 0s describing whether two nodes are connected or not instead of



Graph with two disconnected components.

weighted matrices, which can show the degree of connectivity by a range of values from 0 to 1. Because the similarity matrix, which we will call W, should represent the connections of all the nodes to each other, it should be of size $(n \times n)$ where n is the number of nodes, which in this case is (8×8). Then the row number i and column number j can represent specific nodes, so if the ith node is connected to the jth node in the graph, putting a 1 at w_{ij}, the element in the ith row and jth column in the matrix will indicate this connection. For example, in this graph node 2 is connected to node 1 and node 0, so there will be a 1 placed at position (1, 2), (2, 1), (0, 1) and (0, 2).

In general, the similarity matrix, which we will call W, is an $n \times n$ matrix (where n is the number of nodes) that can be constructed as follows:

$$w_{ij} = \text{similarity score between node } i \text{ and node } j$$

given that

- w_{ij} is the element at row i and column j in W
- A similarity score indicates how similar two nodes are, based on a predetermined metric, that runs from 0 to 1. A larger score indicates a higher degree of similarity.

Similarity graphs are undirected, meaning as long as two nodes are connected, the direction does not matter. Therefore, the resulting similarity matrix should be symmetric, since if nodes i and j have a given similarity score s, w_{ij} and w_{ji} in the matrix will both be s.

2.3 The k Nearest Neighbors Algorithm

We constructed our similarity graph using the k-nearest neighbors algorithm. We decided to choose this algorithm because it is simple and widely used in image segmentation. It is also better than other algorithms like ϵ-neighborhood at connecting data of different scales, or points that have different distances in different regions of space.

Choosing a good k value is also crucial. A k value that is too small can be precise but may lead to overfitting, as the results would be greatly influenced by any kind of noise [3]. However, a k value that is too big can smooth the classification boundaries but may lead to underfitting, as more than necessary neighbors would be considered as belonging to the same group [4]. For our data set, we found that $k = 3$ produced the best results though trial and error.

2.4 The Similarity Matrix

We used our matrix of the row, column, and pixel intensity as our input for the knn algorithm from the NearestNeighbors package. The output of the algorithm was a similarity graph of a vector of s^2 3-element vectors, each containing the pixels closest to the pixel that had the same index as the vector.

```
k = 3
idxs, dists = knn(KDTree(m), m, k)
```

From this similarity graph we obtained the similarity matrix. We created an s^2 by s^2 matrix of zeroes and changed the 0 to a 1 if the pixels at the corresponding indices (the row and column numbers) were similar according to our similarity graph.

```
simMatrix = zeros(s*s, s*s)
for i in 1:size(simMatrix, 1)
    for j in 1:k
        if idxs[i][j] != i
            simMatrix[idxs[i][j], i] = 1
            simMatrix[i, idxs[i][j]] = 1
        end
    end
end
```

3 Extracting the Eigenvalues of the Laplacian Matrix

3.1 The Degree Matrix

First we introduce the concept of a degree matrix, denoted as D. $D \in \mathbb{R}^{n\times n}$ is defined as the diagonal matrix with degrees $d_i, \cdots, d_n$ on the diagonal [1]. The degree of the ith node, d_i is defined as

$$d_i = \sum_{j=1}^{n} w_{ij}$$

To understand this conceptually, the degree of a node, roughly speaking, is how connected it is to other nodes.

The helper function `getDegree` takes in two arguments: `W`, a weighted adjacency matrix, and `v`, the index of a node, and it returns the degree of the node.

```
function getDegree(W, v)
    sum = 0
    for i in W[v,:]
        sum += i
    end
    return sum
end
```

Iterating through all the nodes, we generated the degree matrix of `W` and stored the result in the variable `D`.

```
function getDegreeMatrix(W)
    D = zeros(size(W))
    for v in 1:size(W)[1]
        D[v,v] = getDegree(W,v)
    end
    return D
end
D = getDegreeMatrix(W)
```

3.2 The Laplacian Matrix

For our project, we constructed an unnormalized graph Laplacian, denoted as L, which is defined as

$$L = D - W$$

where W is the undirected, weighted adjacency matrix and D is the degree matrix. To understand spectral clustering, let us first prove the following propositions.

Proposition 1: For every vector $\mathbf{x} \in \mathbb{R}^n$, we have that

$$\mathbf{x}^T L \mathbf{x} = \frac{1}{2} \sum_{i,j=1}^{n} w_{ij}(x_i - x_j)^2$$

where w_{ij} is the element in W at row i and column j and x_i is the ith element of $\mathbf{x}$ [2].

Proof:

First, we show that for all vectors $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{x}^T D\mathbf{x} = \sum_{i=1}^n d_i x_i^2$.

Let $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ and $D = \begin{pmatrix} d_1 & & \\ & \ddots & \\ & & d_n \end{pmatrix}$. Then, we have that

$$\begin{aligned}
\mathbf{x}^T D\mathbf{x} &= \begin{pmatrix} x_1 & x_2 & \cdots & x_n \end{pmatrix} \begin{pmatrix} d_1 & & \\ & \ddots & \\ & & d_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \\
&= \begin{pmatrix} d_1x_1 & d_2x_2 & \cdots & d_nx_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \\
&= d_1x_1^2 + d_2x_2^2 + \cdots + d_nx_n^2 \\
&= \sum_{i=1}^n d_i x_i^2
\end{aligned}$$

Next we show that for all vectors $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{x}^T W\mathbf{x} = \sum_{i,j=1}^n x_i x_j w_{ij}$.

Let $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ and $W = \begin{pmatrix} w_{11} & \cdots & w_{1n} \\ \vdots & \ddots & \vdots \\ w_{n1} & \cdots & w_{nn} \end{pmatrix}$. Then, we have that

$$\begin{aligned}
\mathbf{x}^T W\mathbf{x} &= \begin{pmatrix} x_1 & x_2 & \cdots & x_n \end{pmatrix} \begin{pmatrix} w_{11} & \cdots & w_{1n} \\ \vdots & \ddots & \vdots \\ w_{n1} & \cdots & w_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \\
&= \begin{pmatrix} x_1w_{11} + \cdots + x_nw_{n1} & \cdots & x_1w_{1n} + x_nw_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \\
&= x_1(x_1w_{11} + \cdots + x_nw_{n1}) + \cdots + x_n(x_1w_{1n} + \cdots + x_nw_{nn}) \\
&= \sum_{i,j=1}^n x_i x_j w_{ij}
\end{aligned}$$

Now we are ready to progress to the main proof [2].

$$\begin{aligned}
\mathbf{x}^T L\mathbf{x} &= \mathbf{x}^T D\mathbf{x} - \mathbf{x}^T W\mathbf{x} && \text{(definition of } L\text{)}\\
&= \sum_{i=1}^{n} d_i x_i^2 - \sum_{i,j=1}^{n} x_i x_j w_{ij} && \text{(see above)}\\
&= \frac{1}{2}\left(2\sum_{i=1}^{n} d_i x_i^2 - 2\sum_{i,j=1}^{n} x_i x_j w_{ij}\right)\\
&= \frac{1}{2}\left(\sum_{i=1}^{n} d_i x_i^2 + \sum_{j=1}^{n} d_j x_j^2 - 2\sum_{i,j=1}^{n} x_i x_j w_{ij}\right)\\
&= \frac{1}{2}\left(\sum_{i=1}^{n} d_i x_i^2 - 2\sum_{i,j=1}^{n} x_i x_j w_{ij} + \sum_{j=1}^{n} d_j x_j^2\right)\\
&= \frac{1}{2}\sum_{i,j=1}^{n} w_{ij}(x_i^2 - 2x_i x_j + x_j^2) && \text{(definition of } d_i\text{, } \textstyle\sum_{i=1}^{n} d_i = \sum_{i,j=1}^{n} w_{ij}\text{)}\\
&= \frac{1}{2}\sum_{i,j=1}^{n} w_{ij}(x_i - x_j)^2
\end{aligned}$$

■

Proposition 2: L is symmetric [2].

Proof: Since D is a diagonal matrix, it is symmetric. W is also symmetric by definition. Thus, we know that

$$\begin{aligned}
&\forall i,j \in [n], d_{ij} = d_{ji} && \text{(where } d_{ij} \text{ is the element in } D \text{ at row } i \text{ and column } j\text{)}\\
&\forall i,j \in [n], w_{ij} = w_{ji} && \text{(where } w_{ij} \text{ is the element in } W \text{ at row } i \text{ and column } j\text{)}
\end{aligned}$$

Since $\forall i,j \in [n], l_{ij} = d_{ij} - w_{ij}$ where l_{ij} is the element in L at row i and column j by the definition of L, we have that

$$\begin{aligned}
l_{ij} &= d_{ij} - w_{ij}\\
&= d_{ji} - w_{ji}\\
&= l_{ji}
\end{aligned}$$

Thus, L is symmetric. ■

Proposition 3: L is positive semi-definite [2].

Proof: To show that L is positive semi-definite, it suffices to prove that for all vectors $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{x}^T L\mathbf{x} \geq 0$ (stated in lecture). From **Proposition 1**, we have that for every vector $\mathbf{x} \in \mathbb{R}^n$,

$$\mathbf{x}^T L\mathbf{x} = \frac{1}{2}\sum_{i,j=1}^{n} w_{ij}(x_i - x_j)^2$$

By definition, we know that $w_{ij} \geq 0$. Additionally, as $x_i, x_j \in \mathbb{R}$, we know that $x_i - x_j \in \mathbb{R}$. Since the square of any real number is nonnegative, we have that $(x_i - x_j)^2 \geq 0$. Thus, the product $w_{ij}(x_i - x_j)^2 \geq 0$. Additionally, any summation with nonnegative terms must also be nonnegative, and multiplying the result by $\frac{1}{2}$, a positive value, generates a nonnegative number. Thus, $\sum_{i,j=1}^{n} w_{ij}(x_i - x_j)^2 \geq 0 \implies \mathbf{x}^T L\mathbf{x} \geq 0 \implies L$ is semi-definite. ■

Proposition 4: The smallest eigenvalue of L is 0, and the corresponding eigenvector is the constant one vector $\mathbb{1}$ [2].

Proof: From **Proposition 3**, we have that L is positive semi-definite. Thus, its eigenvalues are all nonnegative (stated in lecture). Therefore, to show that 0 is the smallest eigenvalue, it suffices to show that it is *an* eigenvalue of L corresponding to the eigenvector $\mathbb{1}$. Note that, by the definition of L, for some row $i \in [n]$,

$$\begin{aligned}\sum_{j=1}^{n} l_{ij} &= \sum_{j=1}^{n} d_{ij} - \sum_{j=1}^{n} w_{ij} \\ &= d_i - \sum_{j=1}^{n} w_{ij} && \text{(the only nonzero element in row } i \text{ of } D \text{ is } d_i) \\ &= d_i - d_i && \text{(definition of } d_i) \\ &= 0\end{aligned}$$

Hence, the row sum of L for every row is 0. Thus, we have that

$$L\mathbb{1} = \mathbf{0} = 0(\mathbb{1})$$

Indeed, 0 is an eigenvalue of L corresponding to the eigenvector $\mathbb{1}$. ■

Proposition 5: Let G be an undirected graph with nonnegative weights. The multiplicity k of the eigenvalue 0 of L is the number of connected components $A_1, \cdots, A_k$ in the graph, and the eigenspace of eigenvalue 0 is spanned by the indicator vectors $\mathbb{1}_{A_1}, \cdots, \mathbb{1}_{A_k}$ of those components [2].

First, let us define some terms.

- A *connected component* is a subset of all the nodes A that can be joined by a path such that all intermediate points are also in A, but there are no paths from any element in A to something that is not in A [2].
- An *indicator vector* $\mathbb{1}_A = (x_1, \cdots, x_n) \in \mathbb{R}^n$ is a vector where $x_i = 1$ if the ith node is in A (defined as above) and $x_i = 0$ otherwise [2].

Now we are ready to prove the proposition.

Proof: Suppose we have k connected components. (Note: this k is unrelated to the k value involved in the k-nearest neighbors algorithm.) Let us order the nodes according to which connected components they belong to. It follows that the adjacency matrix W is a diagonal block matrix. To gain intuition for this, consider a particular connected component with a nodes. That means that for each row corresponding to the nodes in a, the element in the ith column for all nodes i that are not one of the a nodes of the connected component must be 0. Figure 3 provides a nice visual for this concept.

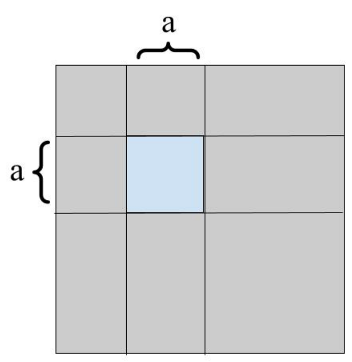


Figure 3: Visualization of diagonal blocks

The entire box represents the matrix. The grayed out regions are filled with zeros, while the blue box has nonnegative values corresponding to the weights of the edges between the nodes in the subset in question. We can see that if we look at each connected component, they will form a similar block along the diagonal. Furthermore, we can also see that each block is also a weighted adjacency matrix on its own with respect to the nodes in that particular connected component.

Since W is a diagonal block matrix and D is a diagonal matrix, L must also be a diagonal block matrix

$$L = \begin{pmatrix} L_1 & & & \\ & L_2 & & \\ & & \ddots & \\ & & & L_k \end{pmatrix}$$

Since the blocks on W's diagonal are weighted adjacency matrices, it follows that the blocks on L's diagonal, $L_1, \cdots, L_k$, are Laplacian matrices on their own as well, where L_i corresponds to the Laplacian of the ith connected component.

From **Proposition 4**, we have that for every L_i, 0 is the smallest eigenvalue (with multiplicity 1) and the corresponding eigenvector is $\mathbb{1}$. Hence, the matrix L has the same amount of eigenvalues 0 as the amount of connected components, and the corresponding eigenvector is the one vector for the nodes in the connected component with zeros elsewhere, which is the indicator vector. ■

3.3 Choosing the k Value

Many attempts have been made to automate the process of choosing the proper k value (number of clusters). For our project, we used the spectral gap heuristic. The spectral gap is defined as the first nonzero eigenvalue [1]. From **Proposition 5** we have that if our data was very neatly segmented into perfect connected components, the multiplicity of the 0 eigenvalue corresponds to k, the number of clusters we should use. Unfortunately, real data rarely segments this perfectly. However, if the weight of the connections between one subset of nodes and another is very low, the eigenvalue, while not perfectly 0, will be very close to

0. Thus, by looking at the smallest eigenvalues of L, we can see how many eigenvalues are close to 0 to estimate the appropriate k value we should use.

Figure 4 shows the graph of the eigenvalues of a Laplacian matrix for our circle image data set. We can see that the first three eigenvalues are very close to 0, with a large gap (spectral gap) between the third and the fourth eigenvalue. This indicates that for this particular data set, $k = 3$ would be suitable, which makes sense, as we have 2 circles and a background.

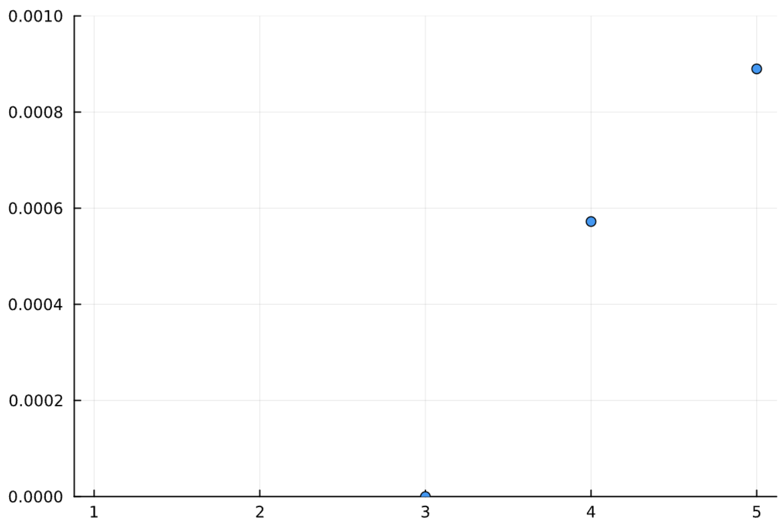


Figure 4: The spectral gap for circle image

The code to generate this graph is as follows:

```
ef = eigen(L)
ef.values
scatter([1:5], ef.values, ylims=(0,0.001), legend=false)
```

3.4 Computing the Eigenvectors

From **Proposition 5**, we can see that the smallest k eigenvectors of L are indicator vectors that encode all the information needed to cluster our nodes. Thus, the next step was to compute the eigenvectors corresponding to the smallest k eigenvalues of L, which we did with the function below.

```
function getFirstKEValues(W, k)
    ef = eigen(Symmetric(W), 1:k)
    return ef.vectors
end
```

We stored these eigenvectors into the columns of the matrix $U \in \mathbb{R}^{n \times k}$.

```
U = getFirstKEValues(L, k)
```

Now we are ready to apply the clustering algorithm.

4 Applying a Clustering Algorithm

4.1 k-Means Algorithm Versus Spectral Clustering

The k-means algorithm is a traditional clustering algorithm that operates on euclidean distance and assumes that clusters are roughly spherical [1]. However, if we imagine a data set that is shaped like rings, k-means will fail, as it will try to assign the points closest to each other to the same cluster (Figure 6), rather than recognizing that the data is partitioned into outer and inner rings, as illustrated below.

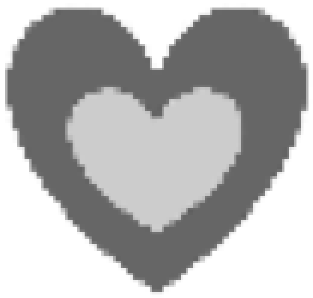


Figure 5: Source image

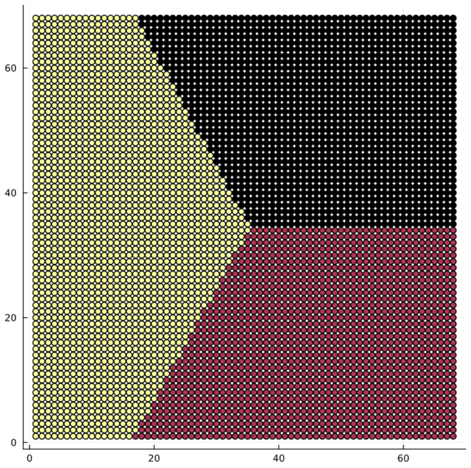


Figure 6: Clustering using the k-means algorithm

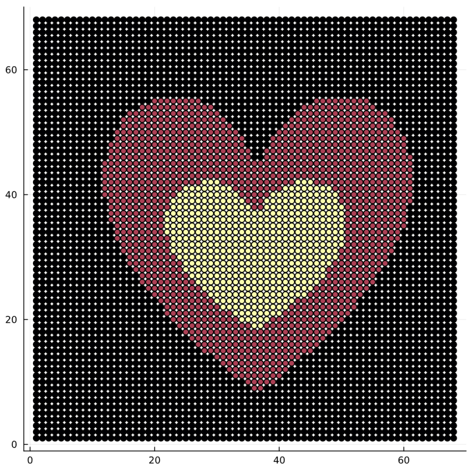


Figure 7: Clustering using our Spectral Clustering algorithm

However, we can see in Figure 7 that spectral clustering takes care of this issue. In spectral clustering, we first transform the data before feeding it into the k-means algorithm, boosting accuracy.

4.2 Running the k-Means Algorithm

To produce our clusters, we used the k-means algorithm on the points represented by the transpose of the row vectors of U and store the result in the variable `R`.

```
R = kmeans(transpose(U), k)
```

This produced labels for all the points, assigning each data point to one of the k clusters.

5 Analyzing the Results

5.1 Plotting the Results

Finally, we produced a scatterpolot of the results with color-coded clusters using the following line of code:

```
scatter(m[2,:], reverse(m[1,:]),
        marker_z = R.assignments, legend = false, aspect_ratio=:1, size=(600,600))
```

With our original data set of two circles in grayscale, we produced a scatterplot that appears to be very similar to the original image. Our algorithm was able to detect the three regions accurately, distinguishing between the two circles and the background.

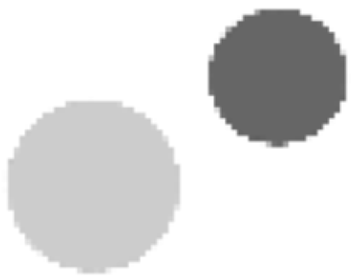


Figure 8: Source image

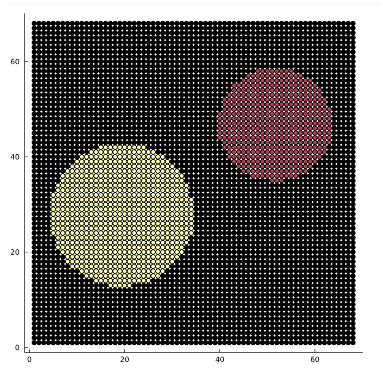


Figure 9: Clustering using the Spectral Clustering Algorithm

Our algorithm also worked well on images with shapes that are not perfectly uniform. Upon close examination, one can notice that each circle has a border of a slightly different color, which makes it a bit harder to detect, as the border pixels would seem less similar to the pixels in the rest of the shape. However, the algorithm was still able to detect that the rings around the circles in the source image below belong to the corresponding shapes.

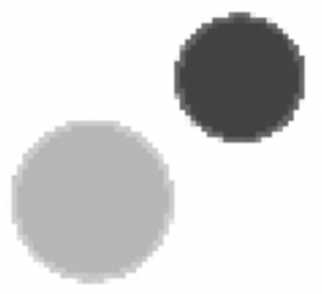


Figure 10: Source image

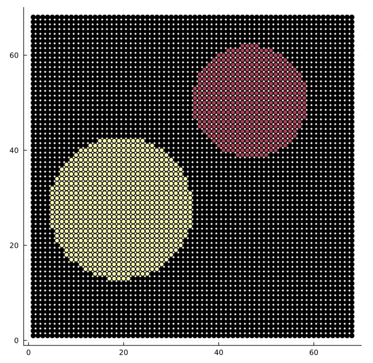


Figure 11: Clustering using the Spectral Clustering Algorithm

Finally, we ran our code on a Mickey Mouse silhouette as a more difficult test since unlike a circle, the mickey mouse shape is irregular. In the end, it was still able to produce successful results.

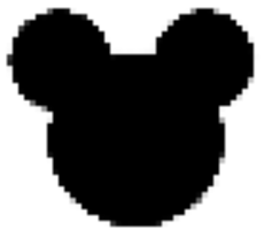


Figure 12: Source image

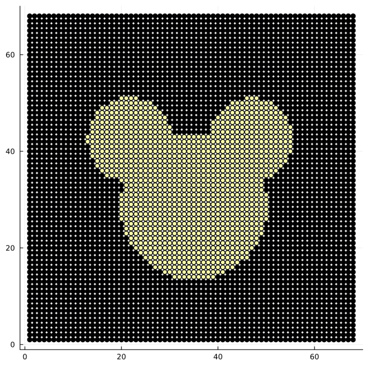


Figure 13: Clustering using the Spectral Clustering Algorithm

5.2 Closing Remarks

Admittedly, our project explores image segmentation at a very rudimentary level. Some possible next steps include optimization, perhaps by partitioning the image into regions and running our algorithm on each partition. That way we could run our algorithm on larger data sets, which are more representative of actual images. Additionally, we could employ algorithms such as the Elbow Method or the Silhouette Method to automate finding the optimal k value for the number of clusters. Furthermore, we could adapt our similarity matrix to take into account of all colors, not just grayscale images. Nevertheless, our project produced satisfactory results on our sample test cases, albeit simple ones.

6 Appendix: The Completed Code

```
using Pkg, Clustering, Plots, Images, NearestNeighbors, LinearAlgebra

# Constructing the similarity matrix
img = load("image.jpg")
new_img = imresize(img, ratio=1/10)
s = size(new_img)[1]
m = zeros(3, s*s)
for i in 1:s
    for j in 1:s
        m[:,s*(i-1)+j] = [i, j, red(new_img[i, j])]
    end
end
k = 3
idxs, dists = knn(KDTree(m), m, k)
simMatrix = zeros(s*s, s*s)
for i in 1:size(simMatrix, 1)
    for j in 1:k
        if idxs[i][j] != i
            simMatrix[idxs[i][j], i] = 1
            simMatrix[i, idxs[i][j]] = 1
        end
    end
end
W = simMatrix

# Constructing the degree matrix
function getDegree(W, v)
    sum = 0
    for i in W[v,:]
        sum += i
    end
    return sum
end
function getDegreeMatrix(W)
    D = zeros(size(W))
    for v in 1:size(W)[1]
        D[v,v] = getDegree(W,v)
    end
    return D
end

# Constructing the Laplacian matrix
L = D - W

# Finding k
```

```
ef = eigen(L)
scatter([1:5], ef.values, legend = false)
k = 3

# Extracting first k eigenvalues of the Laplacian matrix
function getFirstKEValues(W, k)
    ef = eigen(Symmetric(W), 1:k)
    ef.vectors
end
U = getFirstKEValues(L, k)

# Running k-means algorithm
R = kmeans(transpose(U), k)

# Plotting the result
scatter(m[2,:], reverse(m[1,:]),
        marker_z = R.assignments, legend = false, aspect_ratio=:1, size=(600,600))
```

References

[1] Fleshman, W. (2019, February 20). Spectral Clustering: Foundation and Application. Medium. Retrieved December 7, 2022, from https://towardsdatascience.com/spectral-clustering-aba2640c0d5b

[2] Luxburg, U. von. (2007, August 22). A Tutorial on Spectral Clustering. Springer Science+Business Media.

[3] Supervised learning: Introduction to classification: K-Nearest Neighbors cheatsheet. Codecademy. (n.d.). Retrieved December 9, 2022, from https://www.codecademy.com/learn/introduction-to-supervised-learning-skill-path/modules/k-nearest-neighbors-skill-path/cheatsheet

[4] Aggarwal, S. (2020, June 8). K-Nearest Neighbors. Medium. Retrieved December 9, 2022, from https://towardsdatascience.com/k-nearest-neighbors-94395f445221

[5] V, K. (2022, September 5). Spectral clustering: What, why and how of spectral clustering! Analytics Vidhya. Retrieved December 9, 2022, from https://www.analyticsvidhya.com/blog/2021/05/what-why-and-how-of-spectral-clustering/

[6] Image segmentation : Tensorflow Core. TensorFlow. (n.d.). Retrieved December 9, 2022, from https://www.tensorflow.org/tutorials/images/segmentation