A Comparison of Image Segmentation Methods

Marshall Grimmett

May 13, 2021

Abstract

This paper provides a comparison of various image segmentation algorithms with more standard methods such as K-means and Spectral Clustering, but also a newer more specific method known as Efficient Graph Based Image Segmentation. Results of sample segmentations are provided to compare the visual difference between them, as well as thorough training over various hyper parameters to improve accuracy. Lastly the algorithms are tested on an unseen set of images and compared to one another, as well as a comparison to multiple benchmarks from other algorithms performed on the same dataset.

1 Introduction

Image segmentation is a computer vision problem that divides an image into separate parts, known as segments, based on similar characteristics. These characteristics can be color, intensity, and pixel location, among others. It plays a pivotal role in object detection, where certain objects in an image need to be separated from other objects or a background. Then the image is classified using various machine learning techniques. Some of the more common and standard methods involve unsupervised clustering, such as K-means and Spectral Clustering. These methods group together pixels by creating compact clusters that minimizes the distance between the center of the cluster and every pixel within. Other methods include graph partitioning, where a graph is computed with the edge weights being a shared characteristic, like color or location. Then the algorithm will attempt to minimize weights within a segment and maximize the weights between segments. One way of achieving this is with Felzenszwalb's Efficient Graph Based Image Segmentation.
Perhaps one of the greatest difficulties is with evaluating whether or not the algorithm made a good segmentation. Usually, the ground truth is decided by hand drawn segmentations from human subjects, but this can create inaccuracies and requires a lot of work to produce a reliable dataset. The dataset introduced in this paper attempts to solve such problems and even provides an efficient benchmarking software. This paper will explore this dataset using the algorithms previously mentioned and demonstrate effective techniques for training, testing, and evaluating them.

2 Algorithms and Methods

2.1 K-means

K-means is a very simple algorithm that is a very effective method of clustering. It is an unsupervised machine learning algorithm that finds the best cluster centers to represent the data. Given a value of k, the K-means algorithm will compute k clusters that minimizes the within cluster sum of squares

$$arg\min_S \sum_{i=1}^{k} \sum_{x \in S_i} ||x - \mu_i||^2 = arg\min_S \sum_{i=1}^{k} |S_i| Var(S_i) \tag{1}$$

Where $S = (S_1, S_2, ..., S_k)$ is the set of clusters for the set of data points $(x_1, x_2, ..., x_n)$ and μ_i is the mean for the set of points in S_i. K-means is an iterative algorithm that starts with a random set of cluster centers and then proceeds as follows.

1. Assign each point to the closest cluster using Euclidean Distance.
2. Compute the mean of each cluster. These become the new cluster centers.

The algorithm is repeated until the cluster centers no longer change or a stopping criterion is reached, since convergence is not guaranteed. There are other iterations of K-means, such as K-means++, which chooses the initial cluster centers in a "smarter" way in order to speed up convergence. It does this by spreading out the initial cluster centers [1]. The algorithm has a runtime of $O(nkdi)$ where

- n is the number of data points.
- d is the number of features (or dimensions).
- k is the number of clusters.
- i is the number of iterations.

2.2 Spectral Clustering

Spectral clustering is very similar to K-means in that it is an unsupervised clustering algorithm, but it is also equivalent to the kernelized K-means problem. In fact, K-means is commonly used during a stage in the Spectral Clustering algorithm. The first step is to compute the similarity matrix, A, for the data points. This measures the similarity between all points in the dataset. Since the dataset consists of images, they can be converted into a graph where the vertices are pixels and edges are the value of the gradient. The gradient is the difference in intensity between adjacent pixels, then a decreasing function of the gradient is taken as follows

$$G' = e^{-\beta G/std(G)} + \epsilon \tag{2}$$

Where G' is the new gradient, G is the previous gradient, $\beta = 5$ and $\epsilon = 1e^{-6}$. The larger β is, the less independent the segmentation becomes, and it gets closer to a voronoi at $\beta = 1$.

The next step is to compute the Laplacian of A using the diagonal matrix D defined as

$$D_{ii} = \sum_j A_{ij} \tag{3}$$

and the laplacian L as

$$L = D - A \tag{4}$$

Lastly, the first k eigenvectors of L are computed and used as features for a clustering algorithm such as K-means.

2.3 Efficient Graph Based Image Segmentation

Felzenszwalb's Efficient Graph Based Image Segmentation [2] is a very accurate method that uses graphs and minimum spanning trees to compute the difference between regions of an image. This can be used to define whether or not a boundary exists between the regions. First, the image is converted into a graph, similar to spectral clustering, except the similarity between pixels can be intensity, location, color, or other features. In scikit-image's implementation, it is unclear what is used, but it is likely a combination of intensity and location. The graph is defined as $G = (V, E)$ where V is the set of pixels and E is the set of edges between pixels. Each edge has an associated weight $w((v_i, v_j))$ where (v_i, v_j) is an edge in E. Then a segmentation S is produced consisting of components (or regions) $C \in S$ that gives a partition of V.

Now a predicate is defined to determine whether a component contains similar pixels and is different from all other components. The internal difference is computed using the minimum spanning tree $MST(C, E)$ of the given component

$$Int(C) = max_{e \in MST(C,E)} w(e) \tag{5}$$

This simply selects the largest edge weight for the MST of the component. The difference between components is then computed as

$$Dif(C_1, C_2) = min_{v_i \in C_1, v_j \in C_2, (v_i, v_j) \in E} w((v_i, v_j)) \tag{6}$$

Which finds the smallest edge weight between the two components. Lastly, the predicate is defined as

$$D(C_1, C_2) = \begin{cases} true & if\ Dif(C_1, C_2) > MInt(C_1, C_2) \\ false & otherwise \end{cases}$$

where $MInt(C_1, C_2)$ is the internal difference in the components C_1 and C_2 defined by

$$MInt(C_1, C_2) = min(Int(C_1) + \tau(C_1), Int(C_2) + \tau(C_2)) \tag{7}$$

where τ is a threshold function defined as

$$\tau(C) = k/|C| \tag{8}$$

and k is the scale used to determine the size of the components.

The algorithm is covered in Felzenszwalb's paper [2]. To summarize the process, the algorithm performs merges of the vertices into larger components based on the predicate defined above. It also worth mentioning that a sigma value is chosen which determines the Gaussian Kernel diameter in order to smooth the image beforehand. The runtime is $O(m\alpha(m))$ where α is the inverse Ackerman's function. However, a sorting of the weights must be performed first requiring an additional $O(m\log(m))$ depending on the sorting algorithm chosen.

3 Experimental Evaluation

3.1 Dataset Review

The Berkeley Segmentation Dataset [3] was used to test the algorithms mentioned in this paper. The dataset comes from the following paper, *A Database of Human Segmented Natural Images and its Application to Evaluating Segmentation Algorithms and Measuring Ecological Statistics* [4]. The dataset consists of 300 images in both color and grayscale. 200 of those images are allowed for training and 100 for testing purposes. These images were selected from 12,000 hand drawn segmentations from 30 different humans. For this paper, the decision was made to split the testing set into 70 images for training and 30 images for testing. This allows the algorithms to train quicker, but more importantly the hand drawn segmentations were only available for the testing set in the form of an image. The training set used .seg files to determine the segmentation, requiring them to be trained manually or converted to images. Both solutions would increase training time significantly.

3.2 Metrics and Evaluation

Originally the algorithms would be evaluated using The Berkeley Benchmark and Boundary Detection code [3]. However, due to outdated software requirements and insufficient experience in debugging C code, it became infeasible. Instead, a similar method was used to produce similar results with Scikit-Image using the Adapted Rand Error defined as [5]

$$V_{\alpha}^{Rand} = \frac{\sum_{ij} p_{ij}^2}{\alpha \sum_k s_k^2 + (1-\alpha)\sum_k t_k^2} \tag{9}$$

Given that S is the computed segmentation and T being the ground truth. p_{ij} is a probability that a pixel belongs to both S and T, where $i \in S$ and $j \in T$. Furthermore, $s_i = \sum_j p_{ij}$ and $t_j = \sum_i p_{ij}$. Also, α is set to 0.5. Similarly, precision can be computed as follows

$$V_{split}^{Rand} = \frac{\sum_{ij} p_{ij}^2}{\sum_k t_k^2} \tag{10}$$

and recall can be computed using

$$V_{merge}^{Rand} = \frac{\sum_{ij} p_{ij}^2}{\sum_k s_k^2} \tag{11}$$

3.3 Experimental Methodology

3.3.1 Experimental Goals

1. To compare segmentation results between K-means, Spectral Clustering, and Efficient Graph Based Image Segmentation.

2. To compare run times of the above algorithms.

3. To compare segmentation results from this paper with other segmentation algorithms listed here [3]

3.3.2 Experimental Method

The code for this project was written entirely in Python 3.8.5 using Jupyter Notebooks. Sci-kit Learn and Sci-kit Image were used for the algorithms and testing, as well as various image processing libraries.

To start off, each algorithm was explored using a single image from the training set. This process was necessary to determine the validity of the segmentations and was helpful in determining initial hyper parameters for further training on the whole set. For each image, the algorithm was compared to the corresponding ground truth image using the Adapted Rand Error to compute the F-score. Lastly, the average F-score was computed. This process was performed for a list of hyperparameters and the F-scores were compared. Testing the images was a similar process.

Furthermore, certain processing was undertaken for each algorithm to produce results that could be ingested by the Adapted Rand Error function to provide reliable F-scores. The first step was handling the ground truth images. These showed only the edges traced out by the human subjects and needed to be filled in. Felzenszwalb's Efficient Graph Based Image Segmentation was performed with $\sigma = 0.2$ and $scale = 300$. This gave a very good segmentation to be used for testing purposes.

For K-means, the image was converted from a 3D array to a 2D array so that clusters were created purely based on the color of a pixel and not location. Location based clustering is possible with other methods, but it creates inconsistent results with K-means. The RGB values were also scaled between 0 and 1. After the K-means algorithm was completed, the image would just need to be reshaped back into a 3D array.

Felzenszwalb's algorithm did not require any preprocessing, since it was designed for image segmentation. Spectral clustering on the other hand required more work. First, the image needed to be scaled down since the size of the image affected runtime tremendously. Next, the image is converted into a graph and a decreasing function of the gradient is used to compute the edges. After the clustering is completed, the image is transformed back into it's original size. Once resized, the image had to be rescaled to values between 0 and 255.

4 Critical Analysis of the Experimental Results

4.1 Comparing Sample Segmentations

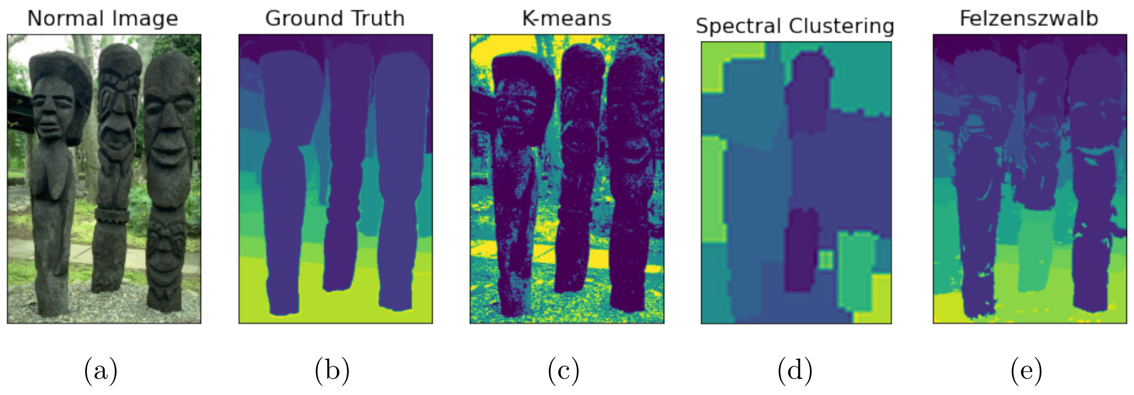


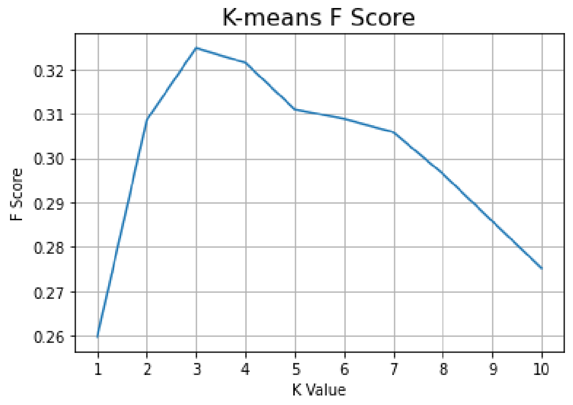
Figure 1: Comparison of Segmentations

In the Figure 1, a sample image was chosen and the segmentations computed for each algorithm. At first glance, K-means seems to have performed well, but upon closer inspection the segments are not separated. The segments are defined by their color with a different color indicating a different segmentation. K-means however, creates clusters solely based on color and not by location, so similar colors are grouped together even if they are far apart or not even connected. This shows the importance that distance plays on producing accurate segmentations. A popular advancement to this is called SLIC which uses K-means on a 5D color space and pixel coordinates.

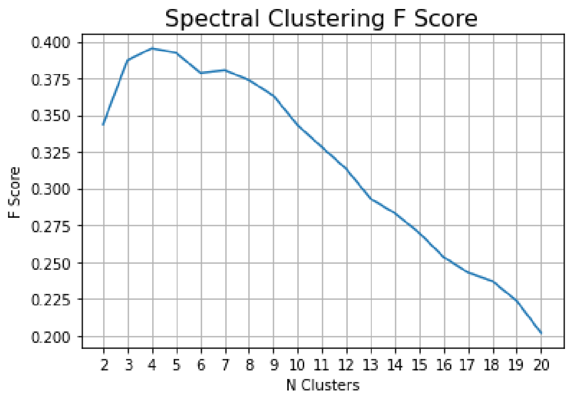
Spectral clustering, performed well in creating segments based on distance, but fails when longer segments are needed. Certain edges can be seen in the segmentation, but they are broken up by other segments, leaving inaccurate results. Felzenszwalb performed the best of the three, but smaller segments within larger ones as well as oversegmentation leads to inaccuracies.

5 Training Results

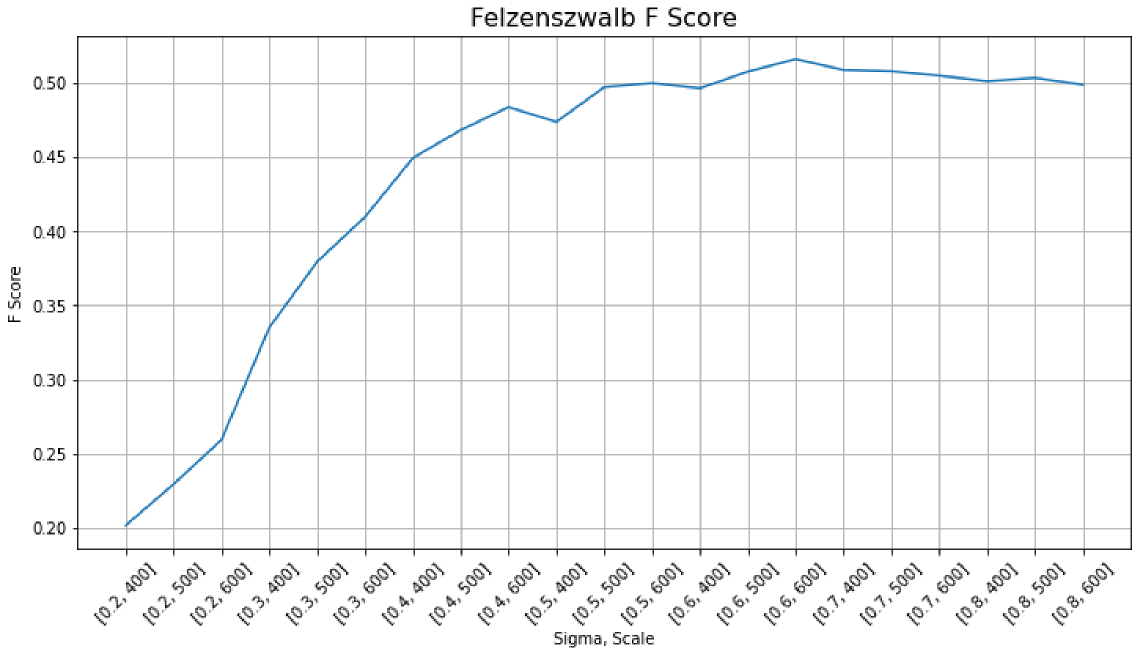
Figure 2 shows the results of training the algorithms on a range of parameters and comparing them to the F-score. For both K-means and spectral clustering it appears that a lower k value provides better segmentations, with larger values leading to oversegmentation. Felzenszwalb was ran with 2 parameters, σ and *scale*, defined in section 2.3. As can be seen from the graph, higher values provides a significantly higher F-score. This is likely due to the fact that a higher scale creates larger segments, which is necessary to match human drawn segmentations.



(a)



(b)



(c)

Figure 2: Training F-scores

5.1 Test Accuracy and Time

Algorithm	F-score	Testing Time (seconds)	Training Time (seconds)
K-Means	0.3156	31.98	1412.43
Spectral Clustering	0.3837	22.12	1155.29
Felzenszwalb	0.4633	9.47	475.29

Table 1: F-scores and Runtimes by Algorithm

In Table 1, the results during training were reflected well in testing, except for Felzenszwalb which was about 0.05 less compared to training. Still, Felzenszwalb performed much better than the other two, with K-means performing the worst. This shows how significant the location of pixel values are when performing image segmentation. When compared to the results from other algorithms [6], K-means and spectral clustering performed worse than the baseline random segmentation. However, it is important to note that the actual benchmark was not performed, so discrepancies are likely when compared to these algorithms. Felzenszwalb performed fairly well, coming close to the 12th algorithm in the list. As for the runtime of the algorithms, Felzenszwalb again performed much better even with a larger range of parameters tested. K-means performed the slowest even with half the range of parameters as the other two.

6 Conclusions

Overall, Felzenszwalb performed better on all fronts compared to the K-means and Spectral Clustering. It also proved to be a competitive algorithm on the Berkeley Segmentation Dataset [3]. This project was also a great demonstration on the importance of pixel location for producing accurate segments and the effect of a graph based approach to segmentation. These results and methods can be further used to train/test other algorithms in preparation for submission to Berkeley's Benchmarking software, since the benchmark takes around 2 hours to complete. This paper also provides decent hyperparameters for each of the algorithms listed, allowing further experimentation to improve them.

References

[1] David Arthur and Sergei Vassilvitskii. K-means++: the advantages of careful seeding. 2007.

[2] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(167-181), 2004.

[3] Pablo Arbelaez, David Martin, and Charless Fowlkes. The berkeley segmentation dataset and benchmark. `https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/segbench/`, 2007.

[4] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proc. 8th Int'l Conf. Computer Vision*, volume 2, pages 416–423, July 2001.

[5] Ignacio Arganda-Carreras, Srinivas C. Turaga, Daniel R. Berger, Dan Cireşan, Alessandro Giusti, Luca M. Gambardella, Jürgen Schmidhuber, Dmitry Laptev, Sarvesh Dwivedi, Joachim M. Buhmann, Ting Liu, Mojtaba Seyedhosseini, Tolga Tasdizen, Lee Kamentsky, Radim Burget, Vaclav Uher, Xiao Tan, Changming Sun, Tuan D. Pham, Erhan Bas, Mustafa G. Uzunbas, Albert Cardona, Johannes Schindelin, and H. Sebastian Seung. Crowdsourcing the creation of image segmentation algorithms for connectomics. *Frontiers in Neuroanatomy*, 9:142, 2015.

[6] Pablo Arbelaez, David Martin, and Charless Fowlkes. Boundary detection benchmark: Algorithm ranking. `https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/segbench/bench/html/algorithms.html`, 2013.