Image Clustering with Noise

Christopher Qian, Hyungjoo Seo

I. Introduction

A. Motivation

There are many potential applications to clustering images. On a fundamental level, we instinctively group different parts of images together. Clustering can provide an automatic way of doing this, which then can be used in a variety of downstream machine learning tasks. However, there are many other reasons, as well. Clustering can also be used as a method to compress an image, by replacing individual pixel values with cluster labels. In addition, clustering can also simply be used to produce aesthetically pleasing results.

In this report, we consider the task of clustering images under noise, in which a certain number of pixels are corrupted. Our goal is to produce clustering results that accomplish one of our desired applications and ideally remove the noise. We consider three types of approaches: 1) Preprocess the image to remove the noise and perform clustering; 2) Perform clustering, then postprocess the image to remove the noise; and 3) Perform clustering in a way that naturally removes noise.

B. Applying K-means Clustering without Noise

First, we demonstrate using K-means to cluster the El Capitan image. To do so, we convert the image into a matrix with NM rows and 3 columns and cluster on the basis of color. Applying K-means with $K = 3, 7$ and 15 demonstrates the various applications of K-means in Fig. 1. We can see that $K = 3$ produces a clustering that reasonably captures a natural way of segmenting the image: the sky, the trees, and the cliff face each form a segment.

However, since the clustering is only on the basis of color, the clusters do not display any spatial consistency. $K = 7$ provides a somewhat artistic interpretation of the landscape. There are enough clusters to show the shadows in the trees and cliff, but few enough to produce a "cartoony" effect. $K = 15$, at first glance, looks like a very faithful recreation of the original image, and takes roughly only 1/3 of the storage space. However, a closer look reveals an inability to handle subtle changes in color. This is most noticeable in the sky, which is uniform in color, but in the original image, there is a clear gradient. In fact, increasing K causes the fidelity to decrease, as then the sky starts to get broken up into unnatural looking clusters.

After performing K-means, we can easily create a soft cluster labels for each pixel in the following way: for each pixel i, define the responsibility it belongs to a cluster k by:

$$r_{ik} := \frac{\hat{\pi}_k \exp\left(-\frac{1}{\sigma^2}\|x_i - \hat{\mu}_k\|_2^2\right)}{\sum_{j=1}^{K} \hat{\pi}_j \exp\left(-\frac{1}{\sigma^2}\|x_i - \hat{\mu}_j\|_2^2\right)}$$

$$\hat{\pi}_k := \frac{1}{n}\sum_{i=1}^{n} I(z_i = k)$$

where $\hat{\mu}_j$ are the cluster means and z_i are the cluster labels and $I(Z_i = k)$ is an indicator function that z_i, label of pixel i, is k. The parameter σ^2 is chosen by hand. The responsibility has the interpretation of the probability that pixel i belongs to cluster k. After computing the responsibilities, we can replace each pixel i with the weighted average of the cluster means:

$$\hat{x}_i := \sum_{j=1}^{K} r_{ij}\hat{\mu}_j$$

The value of σ^2 needs to be chosen; we found that setting it to about 15 times the estimated variance works well. When σ^2 is small, the resulting image looks the same as in K-means. When σ^2 is large, the responsibilities become nearly uniform. However, when plotting, we can scale the values to be between 0 and 1 by dividing by the maximum value. As we increase σ^2, the resulting image looks more and more like the original image in terms of texture, but the color becomes washed out. We note that we do not need to fit a GMM to assign soft labels. Indeed, this is equivalent to fitting a spherical GMM; we found that fitting a more complicated GMM produced worse results.

Fig. 1: K-means clustering on El Capitan. Top left: Original image. Bottom left: K = 15. Top right: K = 7. Bottom right: K = 3.

Looking at the results on the El Capitan image in Fig. 2, we can see that even with only two clusters, the textures of the image are well preserved. Increasing K to 3 results in more accurate colors, and the results don't seem to change by increasing K afterwards. The detail arguably is superior to even K-means with a large number of clusters, being able to capture the subtle changes of color in the sky. However, the color is noticeably duller than the real image. When $K = 7$, we can see a small improvement in the color. Unfortunately, using more than 3 clusters has limited usefulness, as we need to store $K - 1$ responsibilities for each pixel ($K - 1$ since the responsibilities need to sum to 1). Thus, we actually need to use more space than the original image.

C. Different Types of Noise

There are several types of noise that we can add to the images. The most basic type that we consider is uniform noise: we randomly select 18000 pixels and replace them with a random number between 0 and

Fig. 2: Clustering with soft labels. Top left: K = 7. Top right: K = 3. Bottom: K = 2

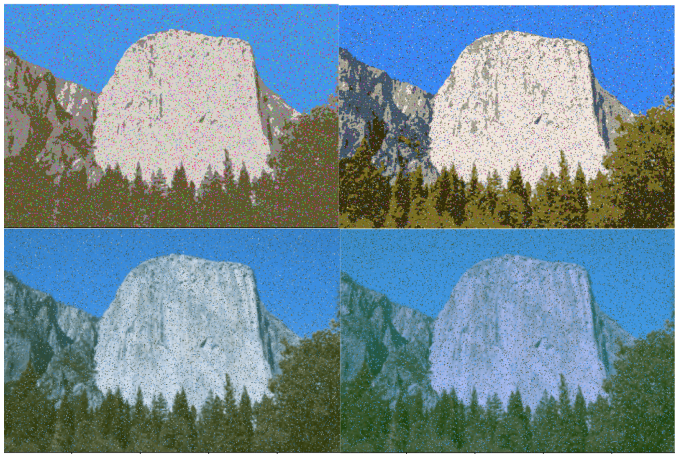

Fig. 3: Effects of noise. Top left: K-means; outlier noise. Bottom left: Soft labels; outlier noise. Top right: K-means; uniform noise. Bottom right: Soft labels; outlier noise.

255 in RGB space (between 0 and 1 if normalized). In Fig. 3, we try standard K-means clustering with $K = 7$ and soft labels with $K = 3$. We can see that performing clustering after uniform noise produces expected results. The clustered image looks basically the same, but there are corrupted pixels speckled in between. Thus, when clustering with uniform noise, we really only have to worry about how to get rid of the noise. On the other hand, we can also consider outlier noise: we randomly select a number of pixels and replace them with colors that are far away from the colors from the image. This, on the other hand, can have adverse effects on K-means. We see that if we include one group of outliers, K-means with $K = 7$ does mostly fine, although some of the colors are pinker than normal. However, with four groups of outliers, K-means loses the texture distinguishing the trees from the mountain. It appears that several of the cluster means are being taken up by the noise, which causes there to be fewer means for the actual image. Thus, when there are several groups of outliers, we would need to come up with a way to prevent this phenomenon, in addition to denoising. On the other hand, clustering with soft labels just tends to change the color depending on the noise. This makes sense, as when there are only 3 clusters, the color will get changed by the noise, but not the assignments themselves.

In this report, we will focus on uniform noise and will discuss ways

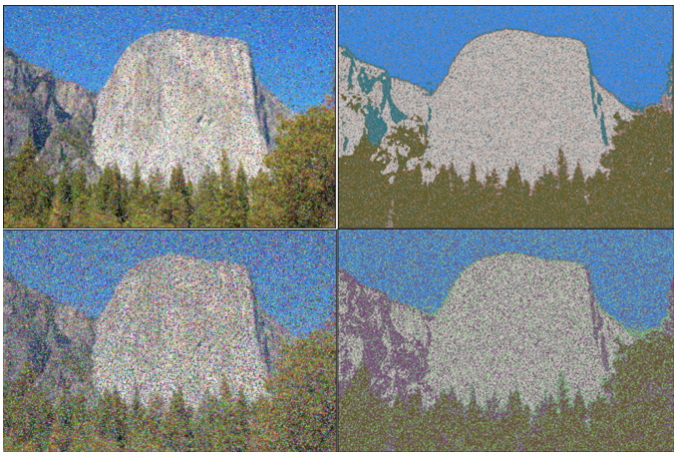

Fig. 4: Impact of NPR on K-means. Top left: 30% NPR. Top right: 30% K-means. Bottom left: 75%., Bottom right: 75% K-means.

to reduce the effect of such noise in clustering problems.

D. Level of noise

While it is shown in previous subsection that K-mean would work just fine with uniform noise, it is not the case when there are more noisy pixels as shown in Fig. 4. To quantify the contribution of noisy pixels over clean pixels (let us call this signal pixels), we defined a parameter that shows a noisy pixel ratio (NPR) as below:

$$\text{NPR } (\%) = \frac{\text{Number of noisy pixels}}{\text{Total number of pixels}}$$

II. Preprocessing to Remove Noise

One way to deal with noisy image is to eliminate the noisy pixels beforehand by preprocessing techniques such as filtering. While there are many different types of supervised or unsupervised denoising techniques suited for all different types of noise [1], [2], here we introduce the efficacy of unsupervised median filtering method which take median among neighboring pixels around a target pixel while sliding the kernel window with the kernel size of L along the entire image x-y space. The logic behind the filter selection is that uniform noise have no correlation among noisy pixels while signal pixels are spatially correlated most of the time. Thus, a corrupted pixel can be recovered by averaging out the RGB values around that pixel. But, instead of taking mean which can get pulled by other noisy pixels around the target noisy pixel, median does better job if kernel size is large enough because adjacent signal pixels are typically clustered in RGB space while noisy pixels are not. Result of median preprosessing is depicted in Fig. 5.

As can be seen, preprocessing can provide fair amount of denoising to recover original image at the cost of blurring effects especially at the edges. Still, there is obvious limitation as noise level increases with this simple filtering methods which necessitates postprocessing afterwards.

III. Postprocessing to Remove Noise

An alternative approach is to perform denoising after clustering. Of course, we can use any of the preprocessing techniques in post-processing as well. However, we also may be able to utilize the clustering information to inform different approaches. In particular, we saw earlier that introducing uniform noise does not adversely affect the clustering. The cluster means, and cluster assignments, look very

Fig. 5: Preprocessing El Capitan. Top left: 30% NPR. Top right: 30% filtered. Bottom left: 75%., Bottom right: 75% filtered.

similar to those of clustering the normal image, except for the corrupted pixels. Assuming that there are not too many corrupted pixels, we can conclude that most of them are relatively isolated, and tend to be surrounded by normal pixels, which have "correct" cluster labels assigned to them. Intuitively, we would say that if a pixel is surrounded on each side by pixels that belong to the same cluster, it should be likely that it belongs to that cluster as well. However, it may also be the case that some pixels actually do belong to the cluster they were assigned to, in spite of their neighbors' assignments. This leads to the following denoising algorithm:

Algorithm 1 Post-processing Denoising

given pixels $x_i \in \mathbb{R}^3$, responsibilities $r_i \in \mathbb{R}^K$ for $i \in \{1, \ldots, n\}$, edge weights w_{st} for $s, t \in \{1, \ldots, n\}$

repeat

  for $i = 1 : n$ **do**

    for $j = 1 : K$ **do**

      set $\log \theta_{ij} = -\frac{1}{\sigma^2}\|x_i - \hat{\mu}_j\|_2^2 - \sum_{l \in \text{nbr}(i)} w_{il} r_{lj}$

    end

    for $j = 1 : K$ **do**

      set $r_{ij} = \frac{\theta_{ij}}{\sum_{k=1}^{K} \theta_{ik}}$

    end

  end

until *stopping criterion*;

return $\{r_i\}$

The edge weights w_{st} determine how much to weigh a pixel's neighbors when updating its responsibility. In our experiments, we set $w_{st} = \frac{J}{|\text{nbr}(s)|}$ for some J. We also need to pick σ^2; we find that a value 10 times as large as that used in regular soft label clustering works well.

We first show results on the grayscale Sailboat image with 2% uniform noise. From Fig. 6, we observe that as we increase J, the resulting image gets blurrier and blurrier, but the corrupted pixels also become less and less noticeable. We see a similar effect in the El Capitan image with 2.5% uniform noise. From Fig. 7, we see that noise in the sky is significantly easier to remove than the noise in the

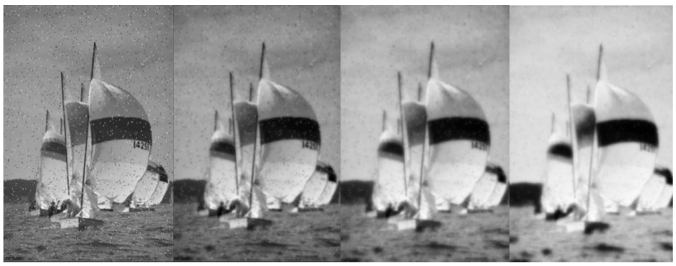

Fig. 6: Postprocessing Sailboat. From left to right: Original image, $J = 1.5$, $J = 1.7$, $J = 1.9$

Fig. 7: Postprocessing El Capitan. Top left: Regular soft labels. Top right: $J = 2.3$. Bottom left: $J = 2.6$, Bottom right: $J = 3$.

mountains. Even with large J, we can still see artifacts of the corrupted pixels on the cliff face. Finally, with the largest J, the image looks similar to a smoothed K-means with $K = 3$.

The results from setting J to be very large suggest another approach to this algorithm. We can perform K-means clustering with a large number of clusters, say, 10, and set J to be very large. Hopefully, the resulting image will look similar to the result of clustering with that number of clusters without noise, and since K is large, the result may look close to the original image. To satisfy the original goal of image compression, we can discard the responsibilities and instead set the cluster assignment of each pixel to the cluster to which it has the highest responsibility. The results show that this approach does not exactly lead to the intended result, as seen in Fig. 8, with 11% uniform noise. With large K, the resulting image is still very smoothed out, resembling an almost impressionist interpretation of the El Capitan image. This approach may have some merit in an artistic sense.

A. A Probabilistic Interpretation of the Post-Processing Algorithm

It can be shown that these update equations can be derived in a principled, probabilistic manner. We assume that the pixel values and their labels are both random variables such that the joint probability of the pixels and their labels is given by

$$\log p(\boldsymbol{x}, \boldsymbol{z}) = \sum_{i=1}^{n} -\frac{1}{\sigma^2}\|x_i - \mu_{z_i}\|_2^2 + \sum_{(s,t)} w_{st} I(z_s = z_t)$$

for some cluster centers $\mu_1, \ldots, \mu_K$, where $w_{st} = 0$ if s and t are not neighbors. Thus, the probability of a particular arrangement of pixels is proportional to how close those pixels are to their cluster centers, and

Fig. 8: Postprocessing with very large J. Top right: $K = 3$. Bottom left: $K = 7$. Bottom right: $K = 15$.

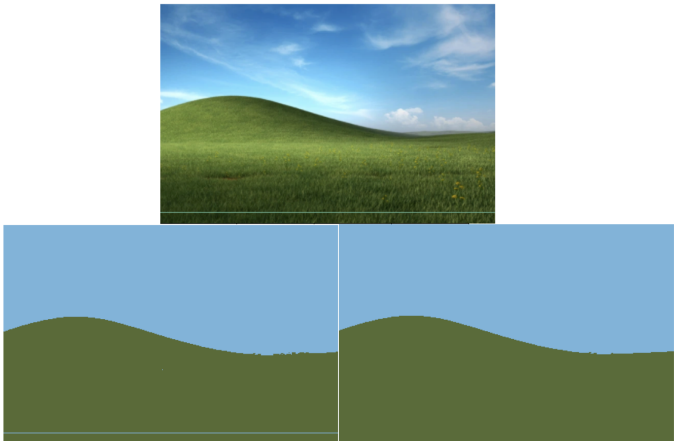

Fig. 9: Bliss segmentation with $K = 2$. Left: K-means. Right: RGB-XY K-means

how similar cluster assignments are to each other. This is equivalent to a GMM with a Potts model as the distribution for the labels. To perform inference using this model, we set the cluster centers to be the ones learned from K-means, and we want to compute the marginal conditional distribution of the labels, that is, compute $p(z_i = k|\boldsymbol{x})$ for each $k \in \{1, \ldots, K\}$ and set these as the responsibilities. Unfortunately, computing $p(\boldsymbol{z}|\boldsymbol{x}) = \frac{p(\boldsymbol{z},\boldsymbol{x})}{p(\boldsymbol{x})}$ is intractable because

$$p(\boldsymbol{x}) = \sum_{\boldsymbol{z} \in \{1,\ldots,K\}^n} p(\boldsymbol{x}, \boldsymbol{z})$$

Thus, we proceed using approximate inference. One popular method of doing so is called the mean-field approximation, in which we assume that

$$p(\boldsymbol{z}|\boldsymbol{x}) = \prod_{i=1}^{n} q_i(z_i)$$

and we solve an optimization problem to find the distributions p_i that minimize the KL-divergence between $p(\boldsymbol{z}|\boldsymbol{x})$ and $\prod_{i=1}^{n} q_i(z_i)$. The usual method for doing so is using coordinate descent, where in each iteration, we update the distributions q_i. This corresponds to the inner for loop in the algorithm. The derivation is analogous to that of using a mean-field approximation where the labels have an Ising model as their distribution, which can be found in [3], pages 737-739.

IV. Joint Denoising and Clustering

A. Clustering in RGB-XY Space

Finally, we consider the approach of clustering in a manner that automatically removes noise. A simple approach is to augment the data by appending the coordinate information. We then perform K-means clustering in this five dimensional RGB-XY space. After estimating the cluster centers and cluster assignments, we replace the RGB value of each pixel with the RGB portion of the cluster assignment. Formally, we solve the optimization problem:

$$\min_{z_i,\mu_i} \sum_{i=1}^{n} (x_{iR} - \mu_{z_i R})^2 + (x_{iG} - \mu_{z_i G})^2 + (x_{iB} - \mu_{z_i B})^2 + \left(\left(\frac{1}{\gamma}\right) x_{iX} - \mu_{z_i X}\right)^2 + \left(\left(\frac{1}{\gamma}\right) x_{iY} - \mu_{z_i Y}\right)^2$$

where γ is a tuning parameter that determines how much to weigh the XY dimension.

The intuitive justification for this procedure is as follows: consider, for example, an image with two clusters, where the top half is one cluster, and the bottom half is another cluster. Ideally, when runnning $K-$means with $K = 2$, we would cluster the top half as one cluster and the bottom half as another. However, when we add noise, some of the pixels in the top half will be closer in color to the bottom half, so they will be clustered incorrectly, and vice-versa. By adding in the XY-dimension, some of these corrupted pixels in the top half will be close in color to the bottom half, but will be closer in distance to the top half. Depending on how much we weigh the position, these corrupted pixels may then get correctly assigned to the top half.

We test this hypothesis on the Bliss image, which was selected specifically because it has two clusters: the sky, and the grass, which roughly comprise the top half and bottom half of the image. In the zero-noise case, we see that including the positional information will create clusters on the basis of position and color. In the $K = 2$ case, it may be more suitable for segmentation, because if there are pixels in the grass that would get classified as "sky", they will get clustered as grass on the basis of position. We can see the benefit when we add a white stripe to the bottom of the image, as seen in Fig. 9

In Fig. 10, we perfrom RGB-XY clustering with 30% uniform noise. We can see that $K = 2$ does not work well; it barely gets rid of the noise; when γ is decreased too much, it produces a degenerate solution. For $K = 4$, the results are promising. However, we are unable to remove the noise in the boundary between the the sky and the grass. This makes sense because the positional cluster centers will be located in the centers of each cluster. However, for the corrupted pixels near the boundary, their distances to each cluster as not that different. In fact, some of the blue pixels in the grass will be closer to a positional cluster center in the sky than a positional cluster center in the grass. If we further decrease γ, then the image starts to flatten the hill. This highlights the biggest problem with this approach. However, it also has other drawbacks. For example, even in the zero-noise case, RGB-XY clustering performs very poorly on the Horseshoe Lake image in Fig. 11, because one of the clusters is a ring surrounding another cluster. Vanilla K-means is therefore unable to assign a single cluster to that ring. We'll see that more sophisticated approaches, such as spectral clustering, can ameliorate this issue.

When we increase the number of clusters, we see a rather interesting effect. In Fig. 12, we apply RGB-XY clustering to Bliss with 1000 uniform corrupted pixels and $K = 100$. The image starts to be broken up into various honeycomb-like chunks, and we see some denoising.

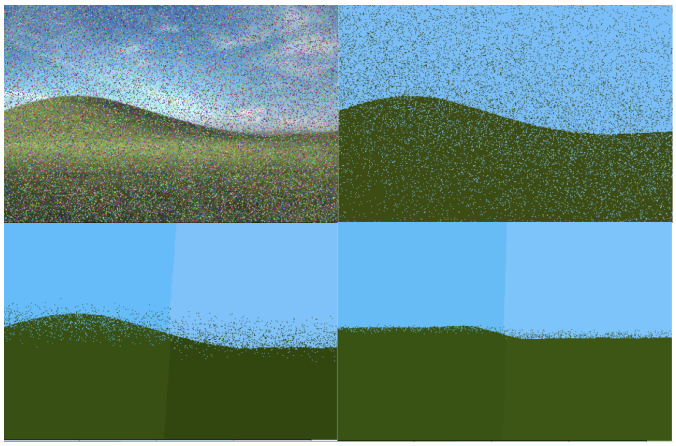

Fig. 10: RGB-XY K-means with uniform noise. Top right: $K = 2$, $\gamma = 295$. Bottom left: $K = 4$, $\gamma = 120$. Bottom right: $K = 4$, $\gamma = 60$.

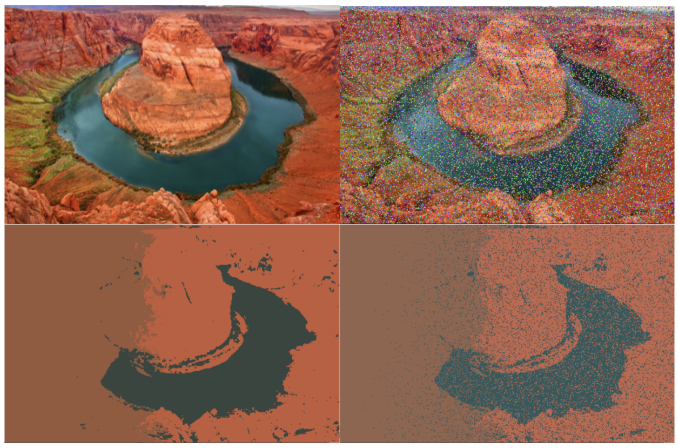

Fig. 11: RGB-XY K-means fails on Horseshoe. Top right: 30% uniform noise. Bottom left and right: $\gamma = 300$.

As we decrease γ, the chunks become more rigid geometrically, and the denoising increases. In particular, we see that in each iteration, certain colors are filtered out. As γ decreases, a wider range of colors get filtered. When γ is set to be sufficiently small, the image becomes tiled by hexagons, producing an interesting artistic effect. This method of denoising is similar to an approach found in the literature based on dividing the image into patches. See, for example, [4]. Unfortunately, we cannot achieve perfect denoising by setting K to be extremely large: when K is too large, the noise starts to get tiled as well.

B. Clustering on Mapped Space from RGB-XY

Another way to apply clustering on RGB-XY space is to perform clustering on a mapped space where mapping provides some desirable properties. In this work, two properties are incorporated: i) in RGB-XY plane, we want neighboring pixels to have far more similarity than far-apart pixels, ii) we want to amplify the discrepancy feature between adjacent pixels and far-apart pixels in RGB-XY plane. Property i) is quite obvious and is already achieved in the approach in previous subsection, but for property ii), a linear combination of respective distances in RGB and XY may not be enough to penalize the impact of far-apart pixels on clustering. Therefore, we propose graph-based spectral clustering approach [5] with nonlinear adjacency matrix W to incorporate aspects of both i) and ii).

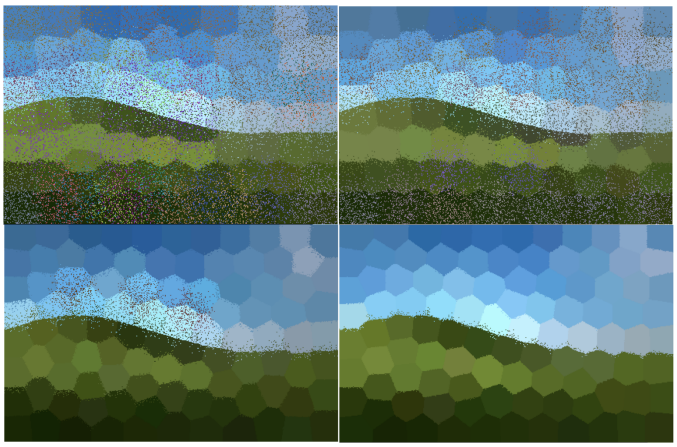

Fig. 12: RGB-XY K-means with $K = 100$. Top left: $\gamma = 80$. Top right: $\gamma = 60$. Bottom left: $\gamma = 40$. Bottom right: $\gamma = 25$.

Here are some steps to perform graph mapping from RGB-XY.

1) Prepare M x N x 3 image with MN pixels containing RGB values ranging from 0 to 255 and vectorize it into MN x 3 array.
2) Map RGB-XY image values into graph domain using adjacency matrix W defined as below:

$$(W)_{ij} = e^{-(\gamma_c|(r,g,b)_i-(r,g,b)_j|^2+\gamma_s|(x,y)_i-(x,y)_j|^2)}$$

where $0 \leq i,j \leq MN$, γ_c represents coefficient of color distance and γ_s represents coefficient of spatial distance.
3) Construct graph Laplacian $L = D - W$ where D is defined as:

$$D = diag(\text{sum of each row of W})$$

Note that graph Laplacian L is symmetric and positive semi-definite matrix which has smallest eigenvalue of 0 with corresponding eigenvector of constant vector $[1...1]^T$. Also, L has MN nonnegative real eigenvalues where the number of first K smallest eigenvalues represent the number of connected components (clusters) of the graph [6].

Therefore, one can perform spectral clustering on eigenspace of nonlinear graph Laplacian matrix L with steps below.

1) Find the K eigenvectors of the K smallest eigenvalues of L.
2) Let U be the MN x K matrix of eigenvectors.
3) Use k-means (or other clustering algorithm) to find K clusters letting each row of U be the data point to be clustered (total MN data)
4) Assign original data point x_i to the j-th cluster if i-th row of L was assigned to cluster j.
5) Find average value of (r,g,b) within each cluster then re-color the picture to get segmented images.

For verification, two aspects were tested with proposed algorithm, segmentation and denoising. First aspect is more sophisticated image segmentation based on nonlinearity to see if i) and ii) properties mentioned above hold. Especially, it is of interest whether this approach can segment water inner ring and red valley in Horseshoe image where K-means + XY had an issue with in Fig. 11. Top and mid images in Fig. 13 show that it successfully segments inner ring from valley structures with the sky depending on selection of K. Since valley in red has more resemblance in mapped space with the water than with the the sky, cluster assignment is prioritized to the sky and the rest of the image when K = 2. And from K = 3 and above, it can be clearly observed that inner water structure is also captured. Also, unlike RGB-XY case in Fig. 11, spectral clustering successfully segmented horseshoe image

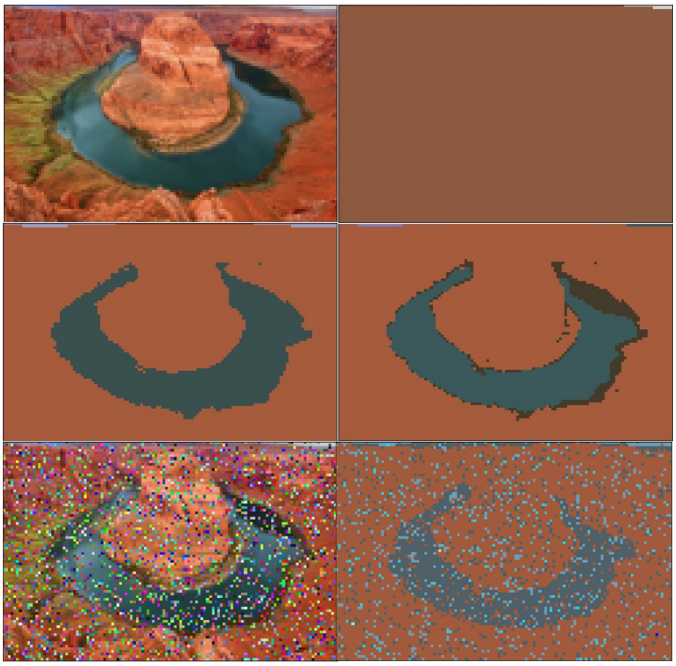

Fig. 13: Spectral clustering on Horseshoe. Top right: $K = 2$. Mid left: $K = 3$. Mid right: $K = 4$. Bottom left: 30% uniform noise, Bottom right: $K = 7$

with 30% uniform noise as shown in the bottom of Fig. 13. Note that, water part does not get prioritized in this image (even some noise pixels take clusters first), thus $K = 7$ was needed to capture the water structure.

Second aspect is robustness against noise when hyperparameters are optimized. Due to the nature of nonlinear radial basis function (RBF), L-space has higher-dimensional than original RGB-XY space. And, for signal pixel's perspective, this mapping enforces both spatial and color similarity while strongly penalizing to zero in W matrix if either dissimilarity is found. Thus, for noise pixels, penalty is given out only the pixel has totally different color from surrounding signal pixels; thus, entire non-diagonal component of W in corresponding noise row is suppressed to near zero. This makes clustering algorithm to prioritize assigning limited number of clusters to signal pixels over **most of** noisy pixels because noise rows of L cannot form a similar block matrix among each other. To demonstrate this, we changed hyperparameters of γ_s and γ_c on Bliss image in Fig. 14 where denoising effect increases as spatial contribution, γ_s, gets boosted. Compared to, RGB-XY's case in Fig. 10, the left-over noise is tracking the curvature of borderline while RGB-XY had almost flat linear line. Also, comparing bottom right images of both figures, it is observed that spectral clustering method does not flatten out the boundaries even when spatial gamma is large enough to get rid of most of noise pixels like it did in RGB-XY.

However, spectral clustering method has a critical drawback which is computational cost because it takes long time to perform i) finding adjacency matrix and ii) eigen-decomposition with MN pixels which can easily go over 10^5.

V. Conclusions

In this report, we have shown several approaches to handling noise when clustering. It turns out that uniform noise can be handled quite gracefully by applying a median filter during preprocessing, even when 30% of the pixels are corrupted. This produces some blurring, but

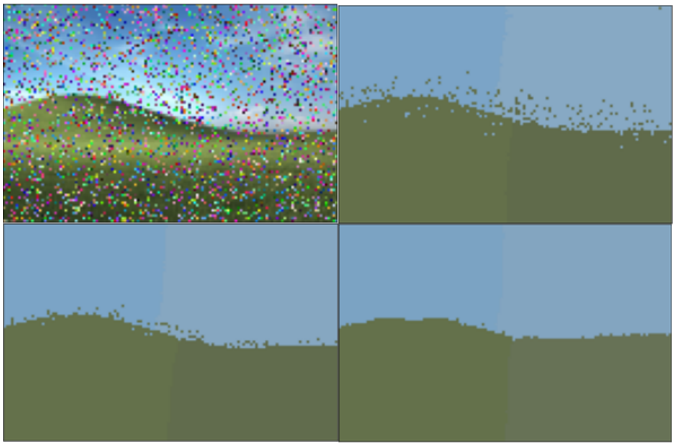

Fig. 14: Spectral Clustering with 30 % Noise, $K = 4$. Top right: $\gamma_s = 10^{-2}$, $\gamma_c = 10^{-4}$. Bottom left: $\gamma_s = 10^{-1}$, $\gamma_c = 10^{-4}$. Bottom right: $\gamma_s = 1$, $\gamma_c = 10^{-4}$.

all of the pixels are smoothed out. The processed image can then be clustered regularly to produce an image close to the clustering of the original image. We also experimented with a post-processing technique to deal with noise after clustering. The results do show show potential to remove noise, but it is not quite as powerful as the median filter, if the goal is to use clustering for compression, as it can only be used with soft labels, and cannot handle as much noise. However, it does have some artistic merit. We also attempted to forgo preprocessing and postprocessing and try methods to jointly cluster and denoise. We found this approach to have the most success in the goal of image segmentation. For the simple Bliss image, RGB-XY K-means is able to remove most of the noise. Spectral clustering can be used to produce better results, and produce sensible results when the clusters are far from Gaussian-like, but is computationally expensive. Future work may be done using these ideas to further improve clustering methods under noise.

References

[1] C.-C. Chang, J.-Y. Hsiao, and C.-P. Hsieh, "An adaptive median filter for image denoising," in *2008 Second International Symposium on Intelligent Information Technology Application*, vol. 2, 2008, pp. 346–350. 2

[2] P. Patidar and S. Srivastava, "Image de-noising by various filters for different noise," in *2010 International Journal of Computer Applications*, vol. 9, no. 4, 2010. 2

[3] K. P. Murphy, *Machine learning : a probabilistic perspective*. Cambridge, Mass. [u.a.]: MIT Press, 2013. [Online]. Available: https://www.amazon.com/Machine-Learning-Probabilistic-Perspective-Computation/dp/0262018020/ref=sr_1_2?ie=UTF8&qid=1336857747&sr=8-2 4

[4] L. Fan, F. Zhang, H. Fan, and C. Zhang, "Brief review of image denoising techniques," *Visual Computing for Industry, Biomedicine, and Art*, vol. 2, no. 1, pp. 1–12, 2019. 5

[5] J. Shi and J. Malik, "Normalized cuts and image segmentation," in *2000 IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, 2000. 5

[6] U. von Luxburg, "A tutorial on spectral clustering," in *Statistics and Computing - Springer*, vol. 17, 2007. 5