Python for Machine Learning (CS477), Spring 2023 K-means Image Segmentation For Noise Reduction In Zebrafish Microscopy Imaging

1st Daniel Blanchette
Department of Computer Science
University of Idaho
Coeur d'Alene, United States
blan5568@vandals.uidaho.edu

I. Introduction

Image segmentation is a popular and innovative field for computer and data science. Research contributing to identifying and isolating image objects contributes to improving many environments for other fields. Prior work on this project required me to isolate microglia cells in zebrafish microscopy images. Using a confocal microscope and a 480nm laser, the bio-luminescent proteins in the microglia are excited and emit lime green coloration to the cells. The original research consisted of finding a threshold method to isolate the cells from the zebrafish image. The issue with this first implementation was that the white pixels with an RGB value of (255, 255, 255) as a tuple conflicted with correctly identifying the cells with a color range from 120 to 255 on the green channel. Often the white pixels were identified as false positives.

A. Task Description

For this project, I chose to apply a popular clustering algorithm known as k-means clustering to reduce image noise. With k-means, I aim to reduce noise from the white channel to pre-process the image set. My initial approach is to test different values of K(centroids) and adjust the centroid placement parameters based on two available options. One uses the traditional method of a random selection of a centroid'(s) locations(k-means) or the improved method of centroid selection(k-means++). I also wanted to research and develop a method to eliminate unnecessary clusters from the results by setting those pixel values to (0, 0, 0). This would act as a bitmask and interact with a cluster based on the layer it belongs to. Due to the image being three-dimensional, I also wanted to see how effective K-means would be in isolating the layers of the image instead of manually setting a threshold to omit certain pixel color values. The total data set consists of 39 images extracted from a .avi file.

II. Methods

A. K-means and K-means++ Custering

Each image was processed using k-means clustering(the traditional algorithm) and k-means plus plus(k-means++). The intention of applying both algorithm implementations was to discern if random centroid selection was the best approach for the model or if using the alternate method of using predictions to select an ideal random centroid position for each cluster would be more beneficial. After some testing, it became evident that the k-means++ implementation would benefit the zebrafish image set more. With traditional k-means, the repeatability of the results was less consistent for centroid selection and image processing. However, k-means++ more often selected similar centroid positions to cluster and, while still randomized, increased repeatability for the data set's output.

B. Model Architecture

In the implementation, a function approach was used for each method to make my code more robust and reusable. For the k-means function, two parameters are passed. The first is `pixel_val` which is the flattened pixel array, and the other is the k value for centroid selection in the algorithm. The other model parameters used for the k-means algorithm are as follows:

```
1 def kmeans_clustering(pixel_val, k =3):
2   """Apply k means clustering to each of the pixel
      values."""
3   criteria = (cv2.TERM_CRITERIA_EPS + cv2.
      TERM_CRITERIA_MAX_ITER, 100, 0.2)
4   """ the last parameter can be adjusted to use
      PP_CENTERS for a slightly more optimized
      centroid calculation if random centers do not
      yield decent results. """
5   compactness, labels, centers = cv2.kmeans(
      pixel_val, k, None, criteria, 10, cv2.
      KMEANS_PP_CENTERS)
6   return compactness, labels, np.uint8(centers)
```

On line three, the defined criteria are used by the OpenCV version of k-means++ for image processing. The parameter `TERM_CRITERIA_EPS` defines the epsilon value for the algorithm(accuracy), and the `TERM_CRITERIA_MAX_ITER` sets the number of iterations for the model. Once processed, the values are stored for the compactness of the pixel array, cluster labels, and centroid locations, which the function returns as output.

Once the k-means++ algorithm has been applied and the clusters are determined, a copy of the k-means segmented

image output is created, and a bit mask is applied to the first two clusters. The following code snippet details the process for this part of the application:

```
def imageMask(image, labels, disabledCluster):
  """Generate a masked image by disabling a defined
     cluster"""
  maskedImage = np.copy(image).reshape((-1,3))
  maskedImage[labels.flatten() == disabledCluster] =
      [0,0,0]
  return maskedImage.reshape(image.shape)
```

This portion of the program aims to take n-defined clusters and set the RGB values to black(0, 0, 0).

A block diagram of the program architecture is provided in Figure 1.

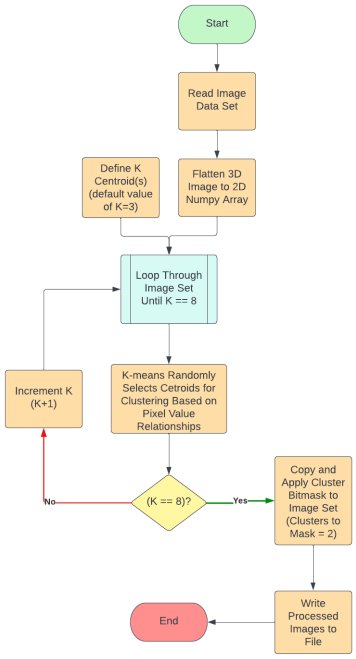

Fig. 1. program architecture: k-means++ segmentation with bitmasking functionality

III. Experimental Results

A. Single Image Test Iterations

k-means++ was applied to a single image from the data set to determine the best k value for image processing. The algorithm performed well when separating individual layers from the three-dimensional image(see Figures 2, 3, and 4).

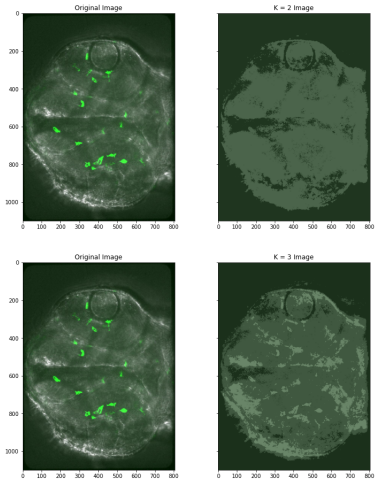

Fig. 2. program architecture: k-means++ segmentation k=2, k=3

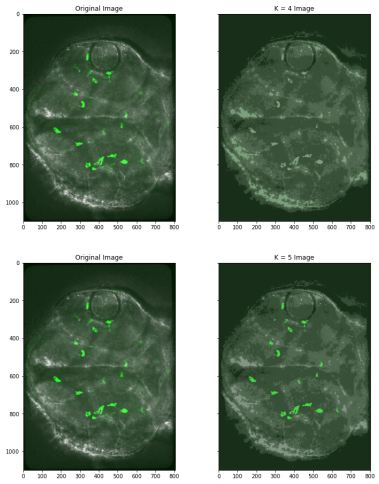

Fig. 3. k-means++ segmentation k=4, k=5

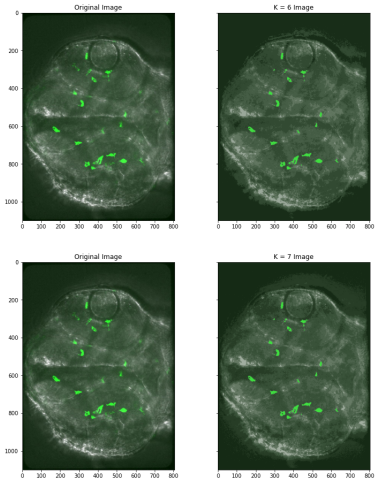

Fig. 4. k-means++ segmentation k=6, k=7

The max k-value tested for this experiment was k=10. I observed that at k=8, the image layers did not appear to change from one k value to another compared to the original image. The next step was to try applying the bit mask routine to the clusters. I tried many values for cluster layers to be altered by the mask and found that labels one and two had the most bearing on image noise from the white channel of pixels(see Figures 5, 6, and 7)

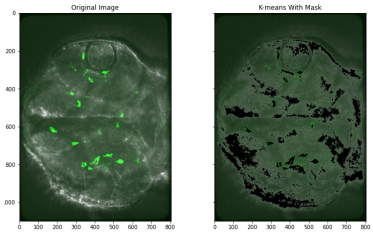

Fig. 5. bitmask incorrectly applied to cluster labels

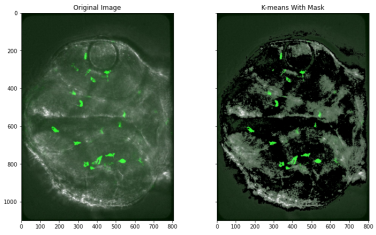

Fig. 6. bitmask incorrectly applied with cell cluster label unaffected

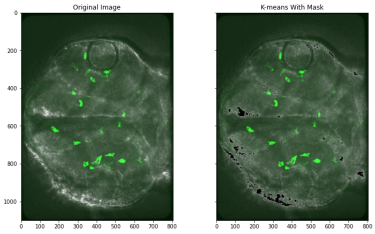

Fig. 7. bitmask applied to correct cluster labels

B. Analysis

The other images resulting in similar output to figures 5, 6, and 7 led me to question how the mask is applied. I knew I could rule out the value of k being the main issue because that determines the number of centroids to be predicted randomly in k-means++. I learned that while k-means and k-means++ are effective algorithms for segmenting image layers in bitmasking, the pixels are subject to the labels the clusters are assigned to. Due to the random nature of this selection process, it became evident that the white channel centroids and cluster labels would be assigned differently with each program run. For clarity, I used the same image for comparison purposes in this report. However, the resulting image data consistently reproduced this issue across all 39 images. Some images would be closer to Figure 7, while others were variations of Figures 5 and 6.

C. Image Assessment

IV. Conclusion

Of the 39 images, 13 (33%) were within a reasonable margin of error to be considered pre-processed for improved segmentation using a threshold technique. The remaining 67% of the images had excessive errors for the k-means layer bitmask application. Based on the experiment and the results, I found this application of k-means++ to be a feasible approach to pre-process an image data set. I propose two methods to improve the program's performance and enhance the results. First, threshold logic could help sort the data into good and poor results. Each data set could then be used by another machine learning model(regression perhaps) on this particular image type to calculate an estimated threshold to isolate the cell objects from the rest of the image. Another approach I didn't have time to experiment with was to use a CNN(Convolutional Neural Network) algorithm for segmentation. The limitation to this at the moment is that the data set is far too small for a CNN to be effective.

A. Discussion

After this experiment, I plan on experimenting with more machine learning models as my internship will start this summer. I aim to gather more data stacks from the research lab I'm working for and apply CNN, U-Net, or another type of segmentation approach to help automate their image analysis tasks in the lab. With a little adjustment, I think noise reduction to the current image set is possible and requires further investigation into methods that can help produce an ideal pre-processed image for further segmentation.

V. References

1) The Python Code Family of Sites. How to Use K-Means Clustering for Image Segmentation using OpenCV in Python, 2023, https://www.thepythoncode.com/article/kmeans-for-image-segmentation-opencv-python. Accessed 05 May. 2023.

2) Digital Sreeni. Image Segmentation using K-means, 2019, https://www.youtube.com/watch?v=6CqRnx6Ic48&t=648s. Accessed 05 May. 2023.

3) Dan Blanchette, Dr. Dianna Mitchell PhD, Dr. Seth Long PhD. Image segmentation of microglia cells in zebrafish, 2022, https://github.com/Dan-Blanchette/INBRE-Internship-2022. Accessed 05 May. 2023