Tyan Trinh

EE 440 Final Project

Autumn 2019

**SLIC K-Means Segmentation**

**Effects/Instructions**

For my final EE 440 project, I did K-means image segmentation. The picture below shows the GUI for my project. In order to upload an image, you hit the bottom left button, then select an image file from your computer. Then, you can drag the two slides to select how many iterations or K means (RGB values) to select.

Since my project randomly chooses RGB values from the original image as superpixel values, these values may not reflect the original image well because some images have millions of pixels and the color selected may not be used that many times. In the same sense, this also affects iterations. When you increase the iterations, the supercluster’s RGB values are converging to a combination that is closest to the original image. It will not improve the rendered image sometimes because some images have at least thousands of RGB combinations, so increasing the iterations doesn’t necessarily increase the quality of the image because it’s just changing the supercluster’s RGB return value to match closer to the image, but when there’s only up to 20 clusters, it doesn’t always help. NOTE: this algorithm sometimes takes a few minutes to run depending on the parameters you select.



**Details of Algorithm/Implementation**

I first flattened the RGB matrices of the original image in order to traverse through all the values easily. Then, the K-means algorithm breaks down into three steps: preprocessing, processing, and postprocessing.

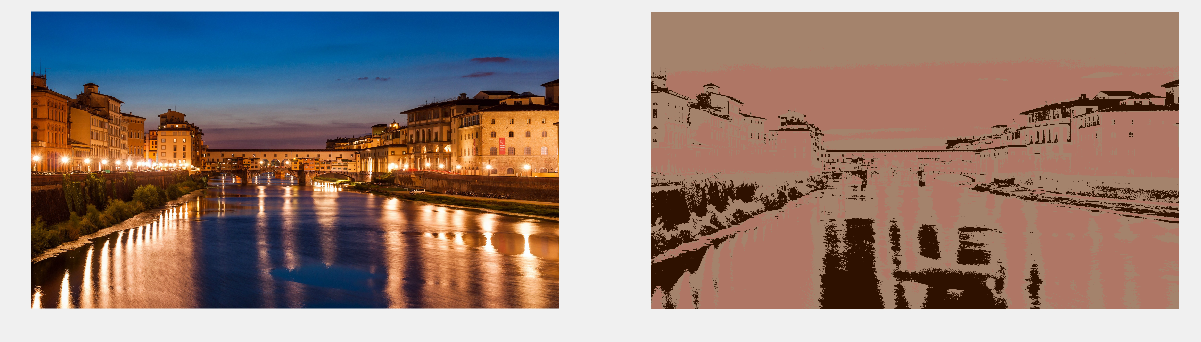
During preprocessing, I chose K-random RGB values from the flattened image/matrix and now, those are my RGB superpixel values. These values get stored in a matrix called SP. Then, I created a new matrix (delta) with all zeros that will will be used for the processing stage.

During processing, for every iteration the user selected, I iterated through the rows of the flattened original image, and for every super pixel (K amount), I took the norm between this super pixel RGB value and the RGB value of the original pixel, and stored this value in a column of delta. Then, I took the minimum of those norms and store a reference to superpixel it will be grouped in. After this, the superpixel gets updated with a new value by taking the mean of all the pixels inside each cluster. The mean and norms are updated for every iteration (which is selected by the user).

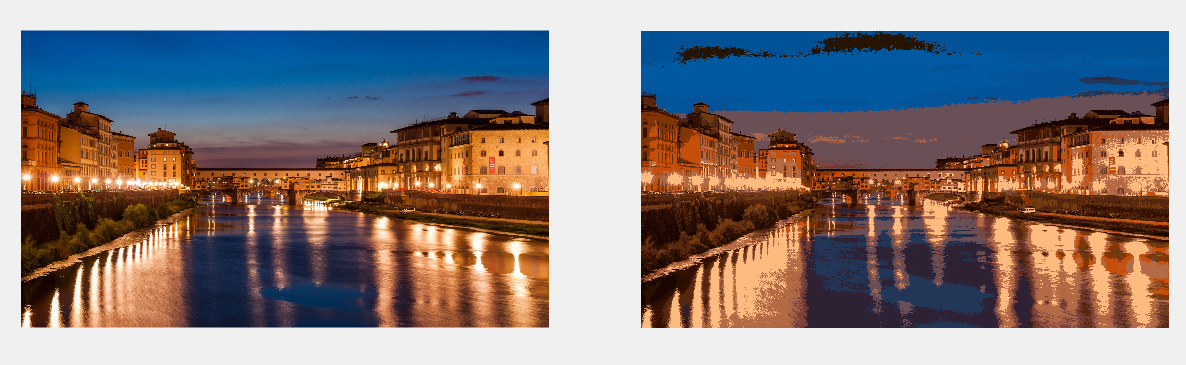
Now, during post processing, the original images RGB intensity values are changed to the RGB values of a superpixel depending on which superpixel the original pixel intensity was assigned to.

**Sample Results**

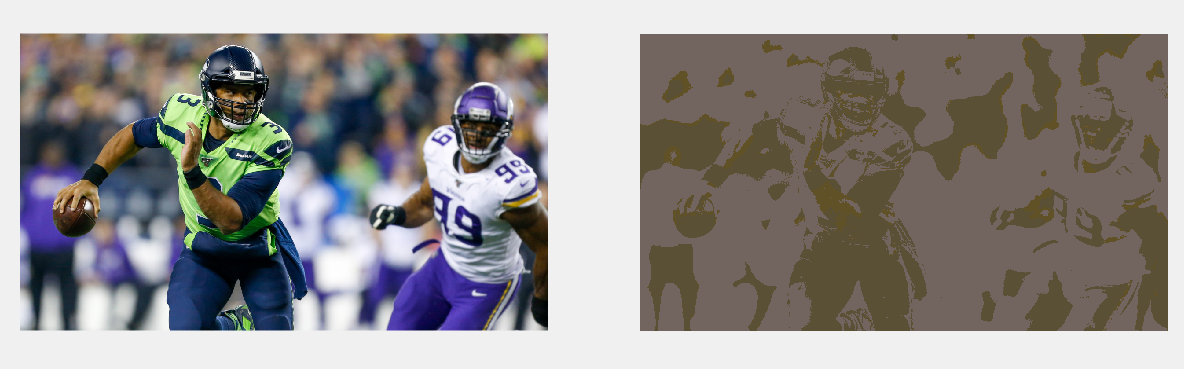
Florence, Italy when you set K = 3.



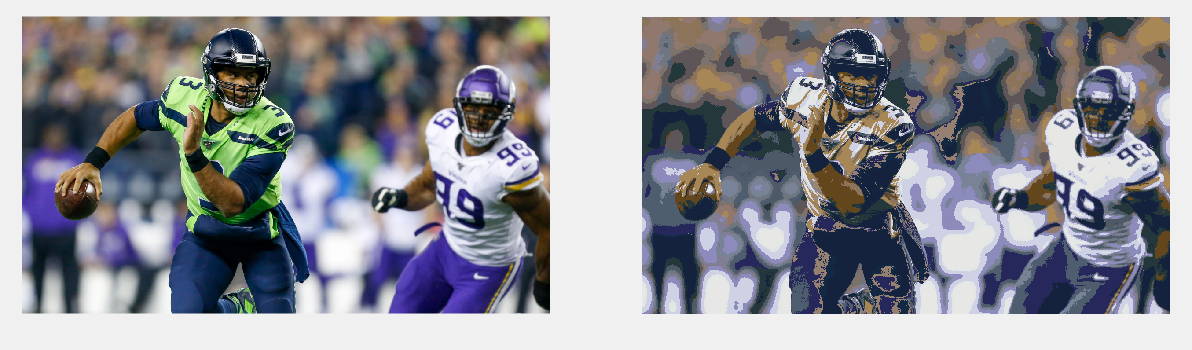
Florence, Italy when you set K = 20.



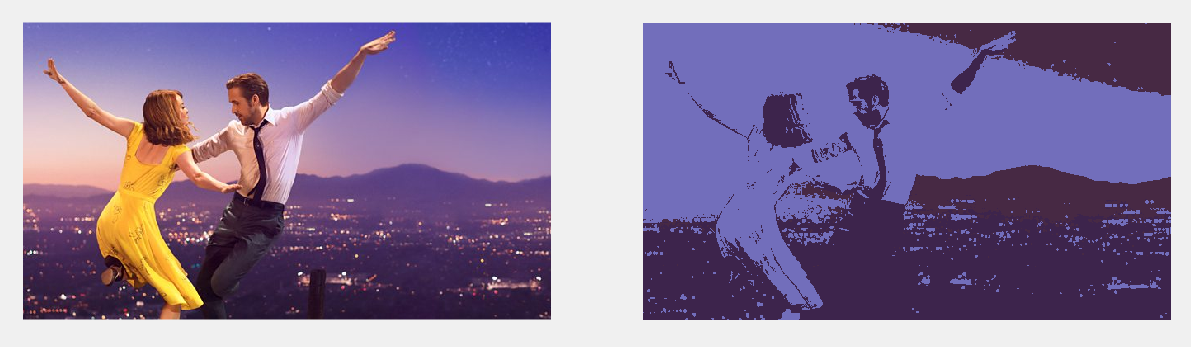
Russell Wilson with K = 3



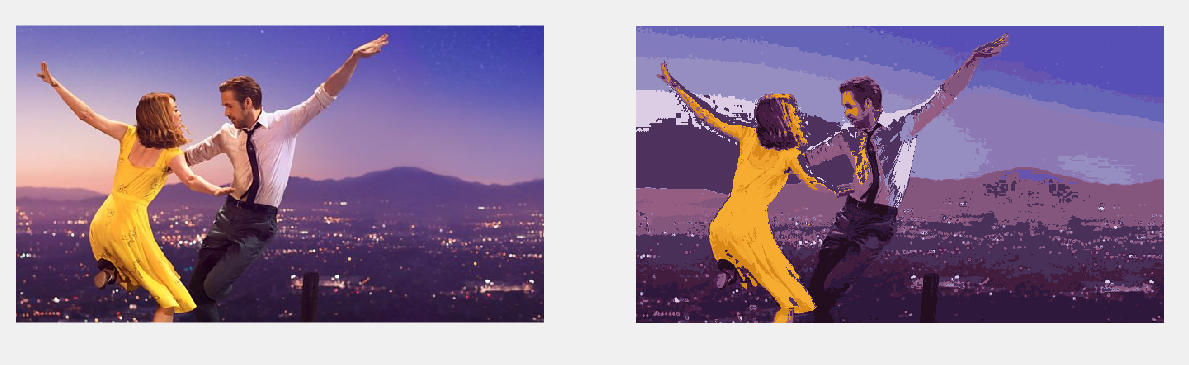
Russell Wilson with K = 20



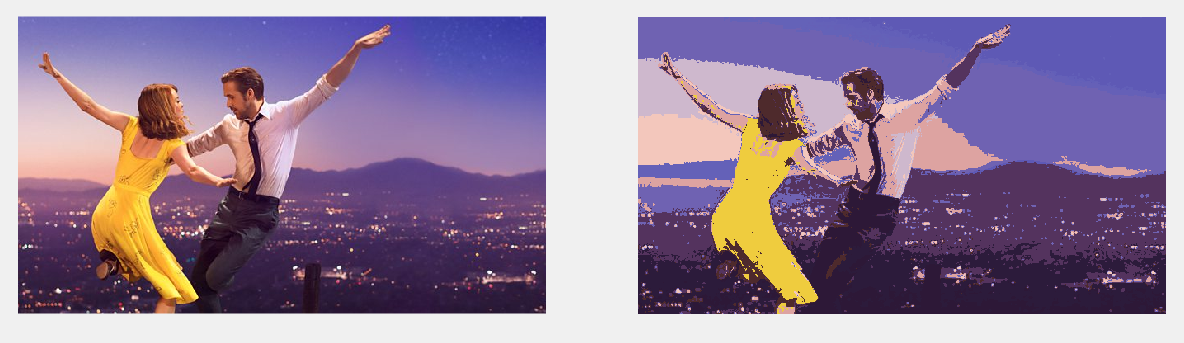
La La Land with K = 3



La La Land with K = 20



Then increasing iterations from 1 to 10. In this case, the yellow and the background’s superpixel colors look more like the original image than when iteration was only 1.



**Reflection/Future**

After implementing K-means image segmentation, I learned the different stages of image processing. In the future, for a more artistic effect, instead of choosing random RGB values from the original image, your superpixels can be chosen by user which would be really interesting.