Image Processing HW5

tags: `2019imageprocess`

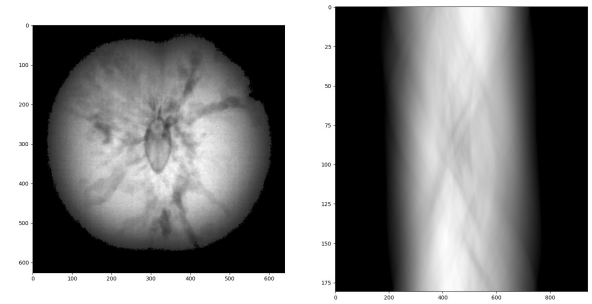
github url: https://github.com/linnil1/2019ImageProcessing/tree/master/hw5 (https://github.com/linnil1/2019ImageProcessing/tree/master/hw5)

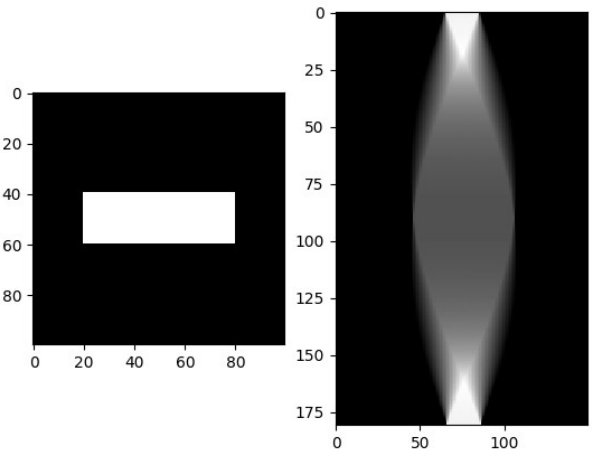
hackmd url: https://hackmd.io/pZ6tFfuXSHO4BG-syObiRQ (https://hackmd.io/pZ6tFfuXSHO4BG-syObiRQ)

Sinogram

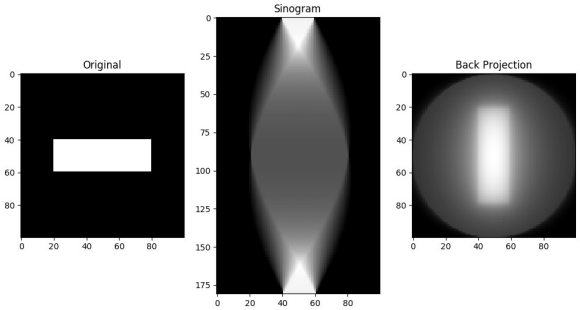
First, get the sinogram from original image.

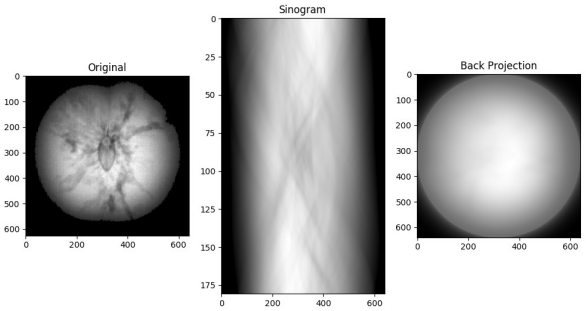
I rotate the image(The same code as affine transform) and do summation along axis.





Then, I repeat each slices along axis and rotated.
Summation all the slice images which is so-called **Back Projection**.





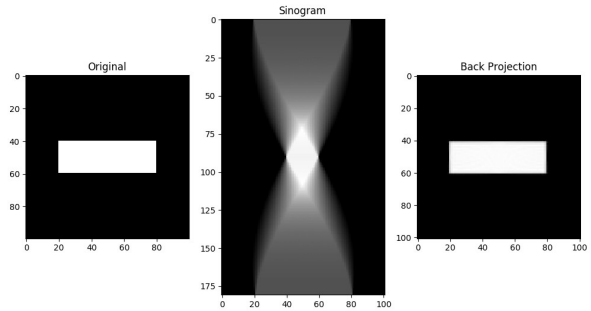
The background is a little bit too bright when reconstructed.

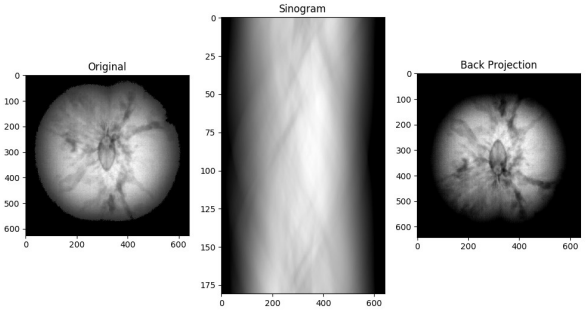
Then, apply Fourier Transform one each slice before the above operations.

Note the inverse equation is a little different, which multiply a ramp function.

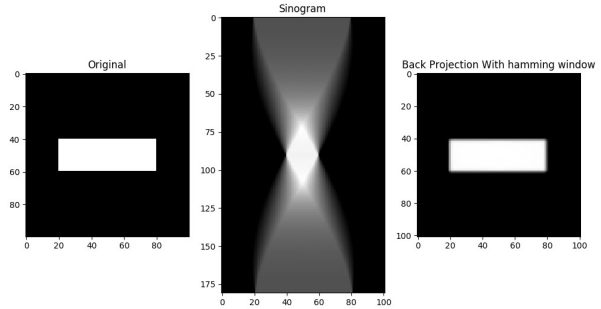
$$\int_{-\infty}^{\infty} |\omega| G(\omega, \theta) d\omega$$

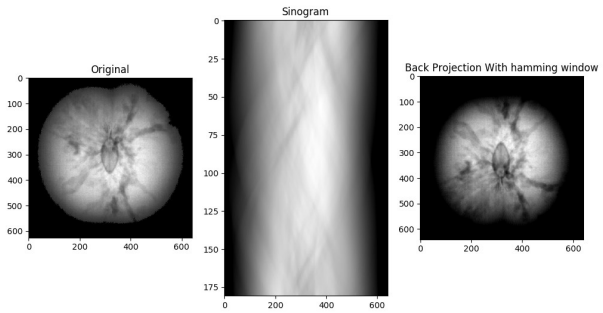
Because the image is discrete, we don't care about whether it is not integrable. The result shown below is awesome.





Try to add hamming window:





It didn't have better performance.

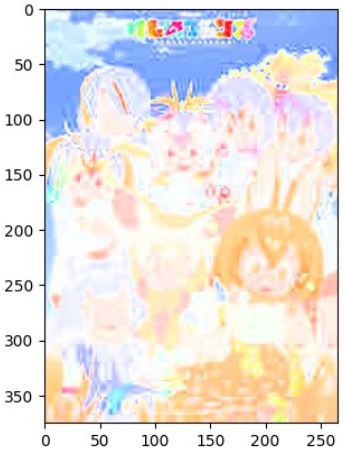
See code `hw5/sinogram.py`

Part1: Color Model Conversion

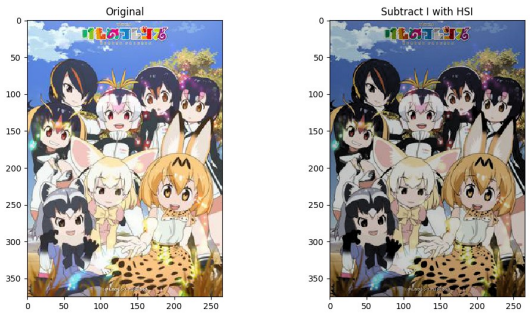
> Design a computer program for color model conversion. Your program should be able to display an image in RGB, CMY, HSI, XYZ, L*ab*, YUV color planes. Check the following web link for more information about color model conversion.
>
> http://www.cs.rit.edu/~ncs/color/t_convert.html
> (http://www.cs.rit.edu/~ncs/color/t_convert.html)

I set RGB as a midway format. Every format will go through RGB and then to the target format.

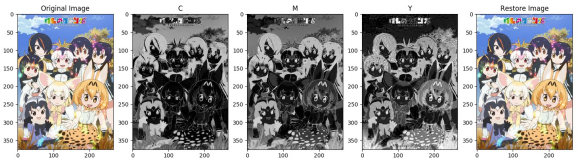
After applying to CMYK, I remove the K layer to stimulate the printer without black ink.

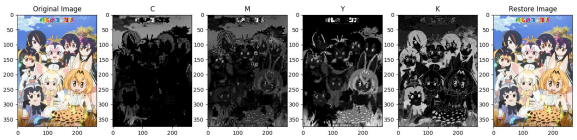


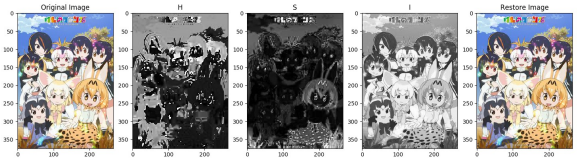
Subtracting 0.2 in I channel at HSI mode is to reduce the brightness.

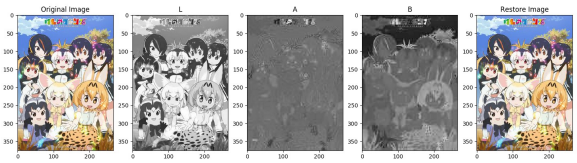


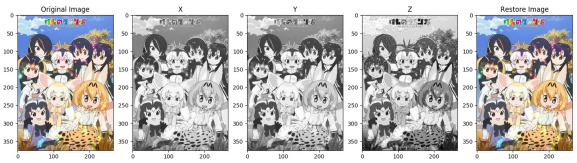
Show all the transformation:

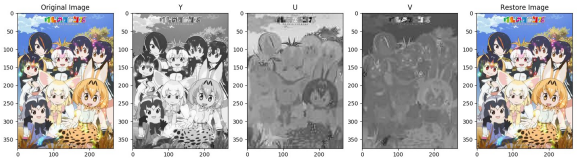












I found that Y vaule in YUV has the same formula in hw2.

I think that implement this is very boring, it is reinventing the wheel.

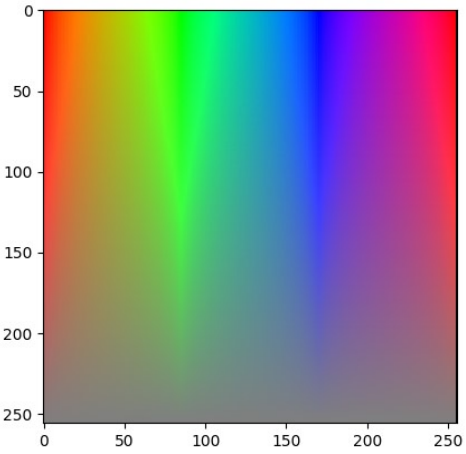
Reference

- https://en.wikipedia.org/wiki/CIELAB_color_space
  (https://en.wikipedia.org/wiki/CIELAB_color_space)
- https://en.wikipedia.org/wiki/YUV#SDTV_with_BT.601
  (https://en.wikipedia.org/wiki/YUV#SDTV_with_BT.601)

Part2: Pseudo-color Image

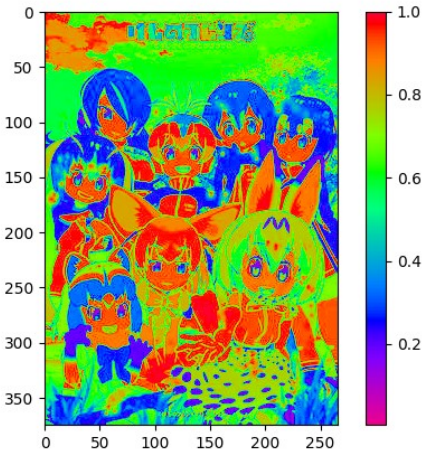
Develop a program to display a grayscale image in pseudo-color. In your program, you need to design an interface to both display the grayscale and pseudo-color images for comparison. You also need to show the color table (color bar) with its corresponding grayscale. Design a user-friendly interface for flexible color table creation. You may learn from popular softwares such as Photoshop or ImageJ for their interface to create a color table.

First create a color picker by listing HSI value. Linearly display HSI as colorbar is more easier than RGB.



Then apply the colorbar to grayscale. Image below is created by two HSI values(Starting color and end color).

```
color_start = [np.pi * 1.8, 1, .5]
color_end = [np.pi * -.05, 1, 0.5]
```



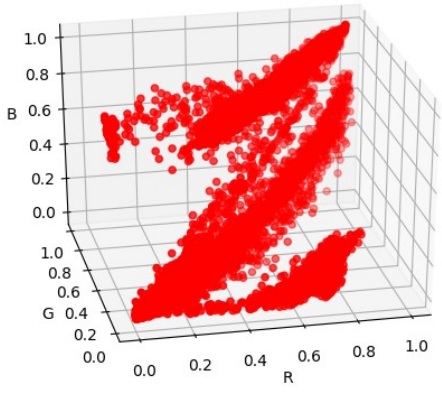
Part3: Color Segmentation

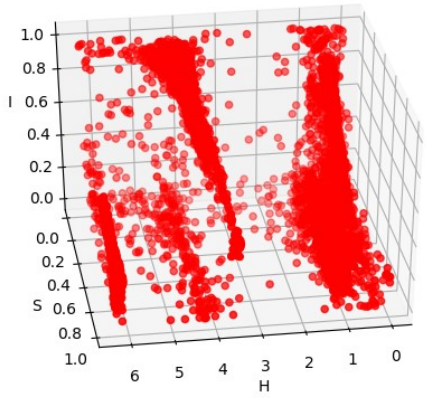
In this assignment, you will practice image segmentation by color clustering, using the k-means algorithm. Your tasks are as follows:

1. Search internet to study the k-means algorithm.
2. Implement a program for image segmentation based on color clustering approach. You may use OpenCV or MATLAB functions for k-means algorithm or you may develop your own function.
3. Test your program with the accompanied images with various levels of complexity. Compare color segmentation results using different k values of the k-means algorithm.
4. Compare the color segmentation results using RGB, HSI, and L*ab* color planes (using k = 2).

First using scatter plot to see whether it has cluster effect base on it's channels(e.g. RGB channel).

Using this image for example(I resize the image to (68, 102) to plotting)





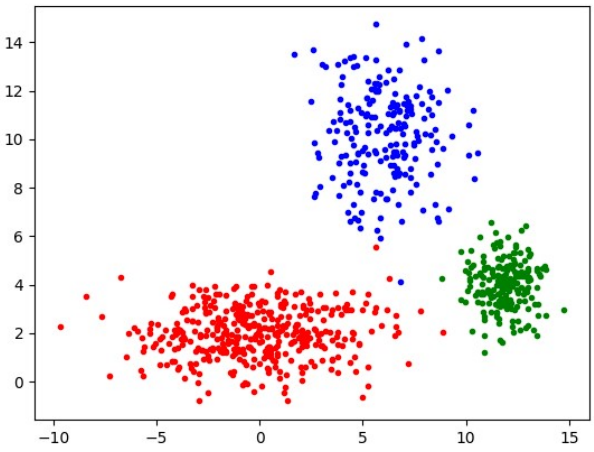
Both of images above show there are four clusters, we can guess that it's red(house), blue(sky), white(cloud), green(tree and grass), gray(Barn maybe?)

Now, try to use k-means to separate them.

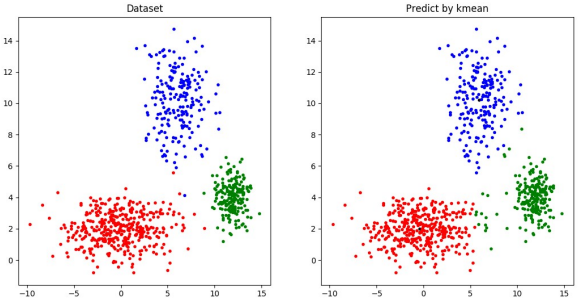
K-means is very easy to implement! Loop this 3 lines then you will find the center of those clusters.

```
dist = np.sum((data - ans[:, None, :]) ** 2, axis=2)
closest = np.argmin(dist, axis=0)
ans = np.array([np.mean(data[closest == i], axis=0) for
```

Let's try on our own easy dataset.

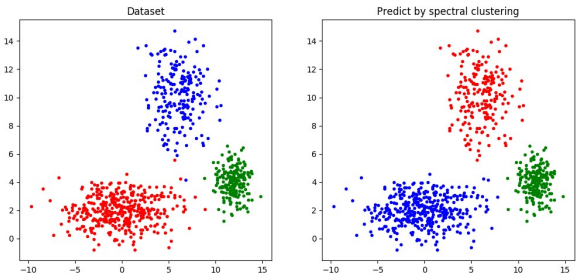


Try applying K-mean. It takes four iterations to complete. The black point is the center point of each cluster.

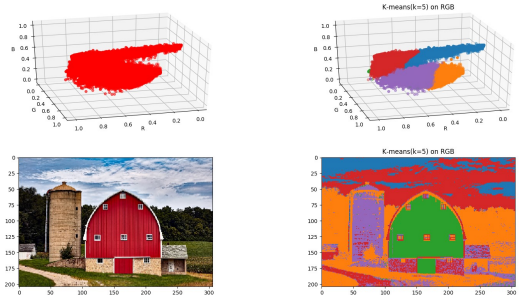


Wow. So easy.

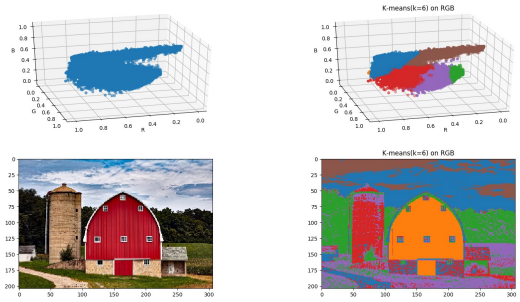
We can see another methods work better than k-means by reference https://scikit-learn.org/stable/modules/clustering.html#optics (https://scikit-learn.org/stable/modules/clustering.html#optics). E.g. spectral clusting, which result is much similar to my dataset in the experiments.



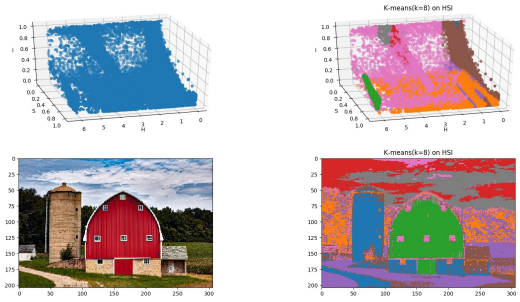
Then apply on the image features. Using K-means



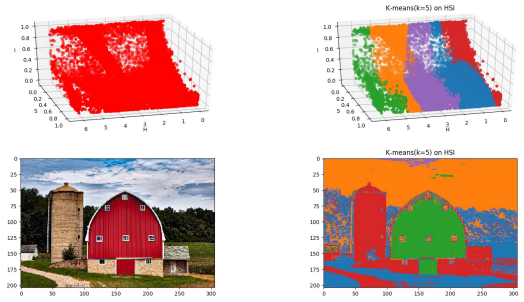
When K=6, I think the result is what we want.



Using gaussian mixture, we should change K value to separate the cloud and the sky.

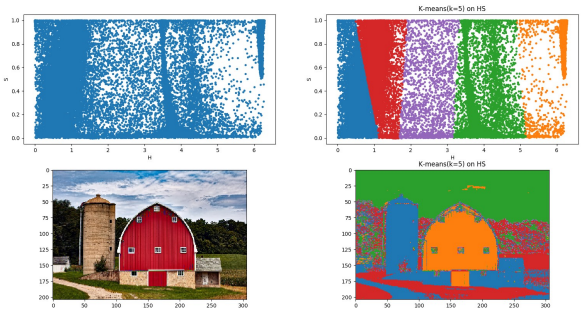


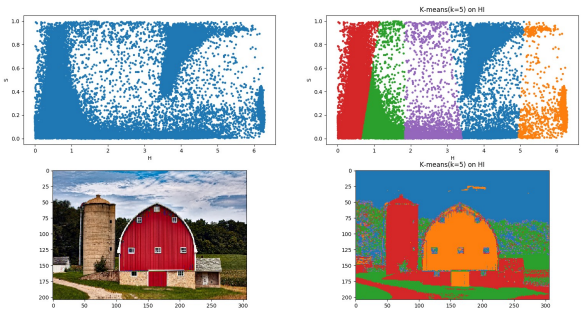
Apply on HSI



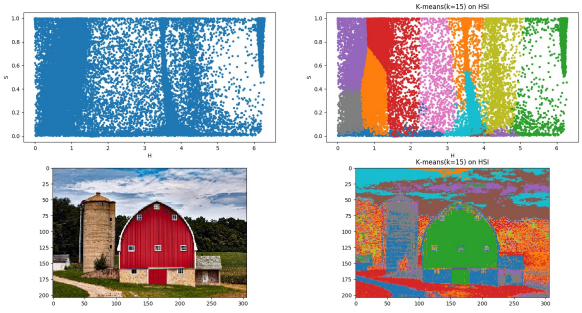
I can't found a suitable k-value to separate grass and forest color by k-mean in HSI.

We can found out that intesity or saturation can be deleted without damaging the result

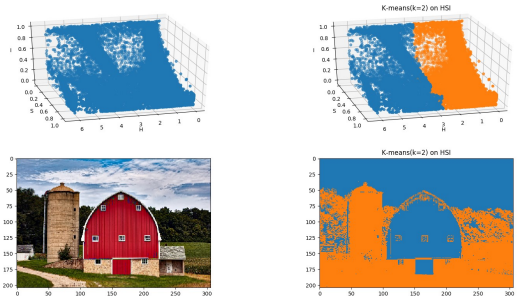




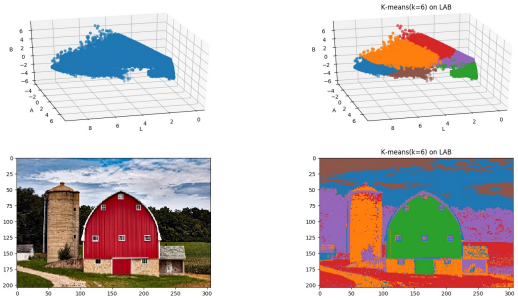
Add more clusters can get more details. When the number of clusters is equal to all the pixels in the image, it'll be identity.



Reduce cluster number will combine very different color into one cluster. When k=2, work similar as binarize the image.



Changing to another space LAB



LAB work very well with k-mean to separate the view I predicted mentioned before. And this contains much less noise than k-means on RGB.

See `hw5/test_kmean.py` for more details.

Program usage

Install on local machine

```
./setup.sh
```

Install by Docker

```
docker build -t linnil1/2019imageprocess .
docker run -ti --rm \
    --user $(id -u) \
    -v $PWD:/app \
    -v /tmp/.X11-unix:/tmp/.X11-unix \
    -e DISPLAY=$DISPLAY \
    linnil1/2019imageprocess python3 qt.py
```

Without QT

You can use command line as a postfix image processor.(Same as HW2)

For example

- `python3 hw5_np.py --read data/HW05-3-03.bmp --colortransform RGB HSI --show`
- `python3 hw5_np.py --read data/HW05-3-03.bmp --kmean 6 --showpseudo 110101 ffffff`

You can use `python3 hw5_np.py --help` to see more.

QT

`python3 qt.py` or `./run.sh`

A dynamic UI image processor.

Note:

- All filter should run under gray scale image.
- The parameters of gaussian is cutoff value not sigma.
- To display the Difference operation in convenience, the code is not only `img1 - img2`. Instead, I use `(img1 - img2) / 2 + 127`.
- Add colormap creation module before using pseudo color module
-
- I using qt built-in `QColorDialog.getColor` as a color picker.

demo Image