Project 3

Boling, Kenneth S.[1]
(1)Earth and Planetary Sciences, University of Tennessee, Knoxville, TN 37996
kboling4@vols.utk.edu
ECE 571

Abstract

Unsupervised learning can be used in a variety of applications, some of which are not immediately apparent. Unsupervised learning is the process whereby objects can be classified based on their similarity to each other without using predefined classes. In this project several unsupervised learning algorithms were used to classify the colors of an image. The classified colors could then be used to create an image with fewer colors than the original true color image. This process has implications for image compression and storage.

Contents

Introduction

In contrast to supervised learning, unsupervised learning algorithms are implemented on data sets without labels or predefined classes. This project used several different unsupervised learning algorithms to classify the pixels in the image below:

Figure 0-1: Original image flowersm.ppm

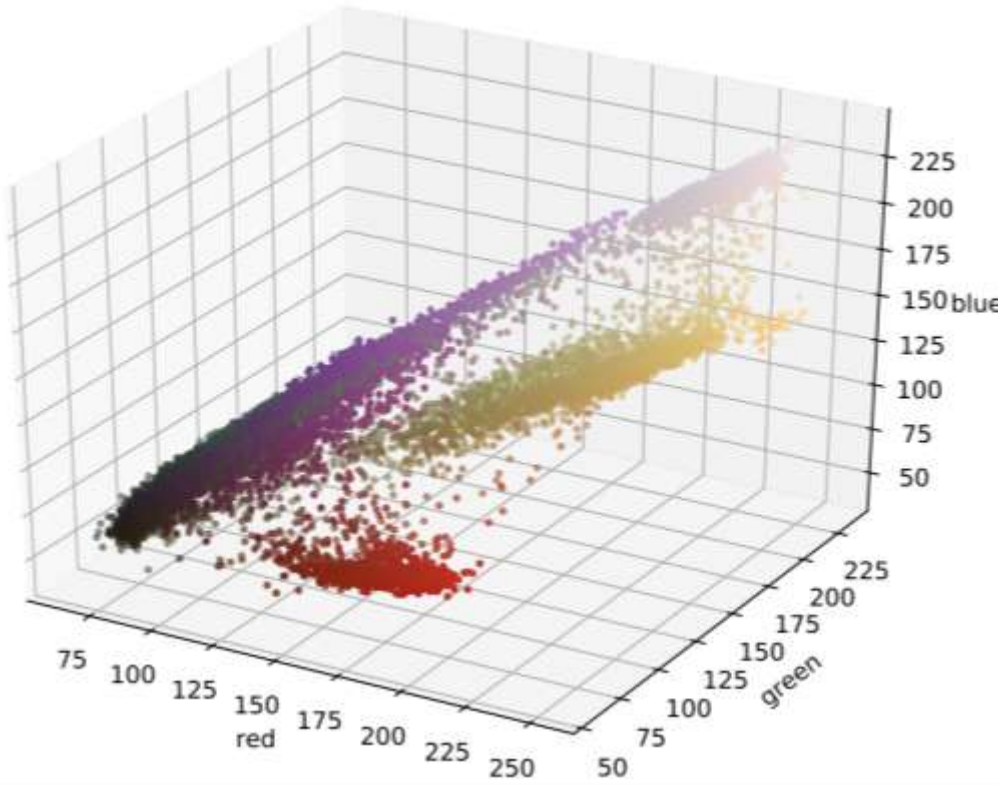
This image is 120x120 pixels and contains 8458 different colors out of 14400 total pixels. The goal of this project was to reduce the total number of colors down to 256 or less.

Methods and Technical Approach

In order to facilitate the use of these clustering algorithms it was necessary to develop a script in Python. This script was written in Python version 3.6.2, using the PyCharm development environment. The script developed to accomplish this task employed the use of several open source Python packages from the *SciPy* library (Jones et al., 2014). The *Pandas* (McKinney, 2010) package was used primarily for the initial data loading, and the *NumPy* (Oliphant, 2006) package was used as the primary tool for data management and processing.

Preprocessing

The .ppm image file was loaded in using the *Pillow* fork of the *Python Image Library (PIL)* package which is no longer in development. The image was then converted into a (120, 120, 3) *NumPy ndarray* with the three dimensions representing the x and y coordinates and the three 8-bit unsigned integer values representing the RGB values for each pixel. The original image contains 8458 total colors out of 14400 total pixels. The Red, Green, and Blue values for each pixel can be used to represent the image in a 3-dimensional space (RGB colorspace).



The pixels from the original image with similar colors form clusters of points. The goal of the clustering algorithms will be to define these clusters. Once these points have been assigned to clusters the colors may be replaced by the values of the cluster centroids. The advantage to this is that a visually identical image can be represented by a smaller number of colors than the original, allowing the image to be stored at a smaller file size. Certain image formats use compression algorithms exploit this to lower the file size of an image.

K-means

The K-means clustering algorithm works by first generating a series of random points within the range of the testing data, these are the centriods. The number of points has to be specified before hand as the value of (k). Each testing data point is then assigned to a class based on the closest centroid, and a new set of centroids are generated at the mean of each of these classes. This process continues until the new centroids generated are equal to those from the previous iteration. Applying a k-means algorithm to an unlabeled testing data set will essentially result in each point being assigned to one of the centriods, which can then be used, in this case, to assign a new color to each pixel within the original image. The drawback of using a k-means algorithm is that the number of *k* classes must be defined beforehand.

Winner-Take-All

The Winner-Take-All Clustering algorithm works in a similar manner to k-means.

Kohonen maps

Kohonen maps work similarly to WTA algorithms.

Means Shift

The means shift algorithm was originally developed by (K. Fukunaga and L. Hostetler, 1975). This clustering algorithm does not require a number of clusters to be identified beforehand.

For this project the *Scikit-Learn MeanShift* function developed by (Comaniciu and Meer, 2002) was adapted to determine the clusters in the image RGB data. The means shift algorithm is useful because it makes no assumptions about the data or the number of cluster in a sample. The only parameter of this algorithm is the bandwidth of the data set, which may be estimated from it. Initialize random seed, and window, calculate center of gravity (the "mean"), shift the search window to the mean, and then repeat until the convergence:

$$x_{t+1i} = m(x_{ti})$$

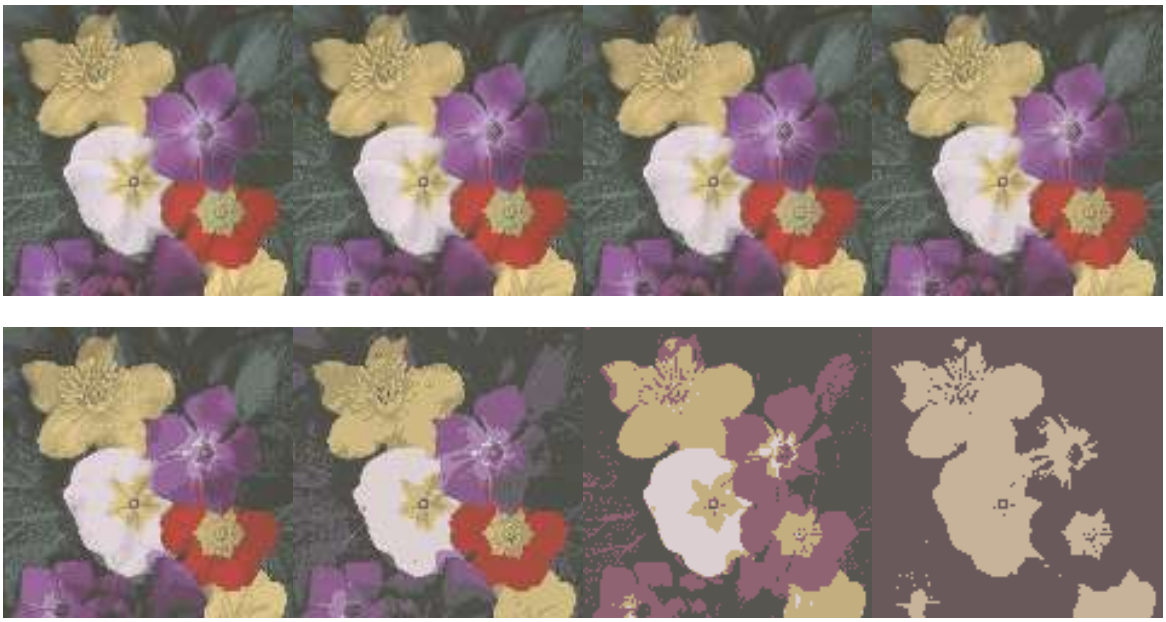
Experiments and Results

The results of each of the algorithms used to find the clusters in the image data are shown for each type of algorithm below:

K-means Results

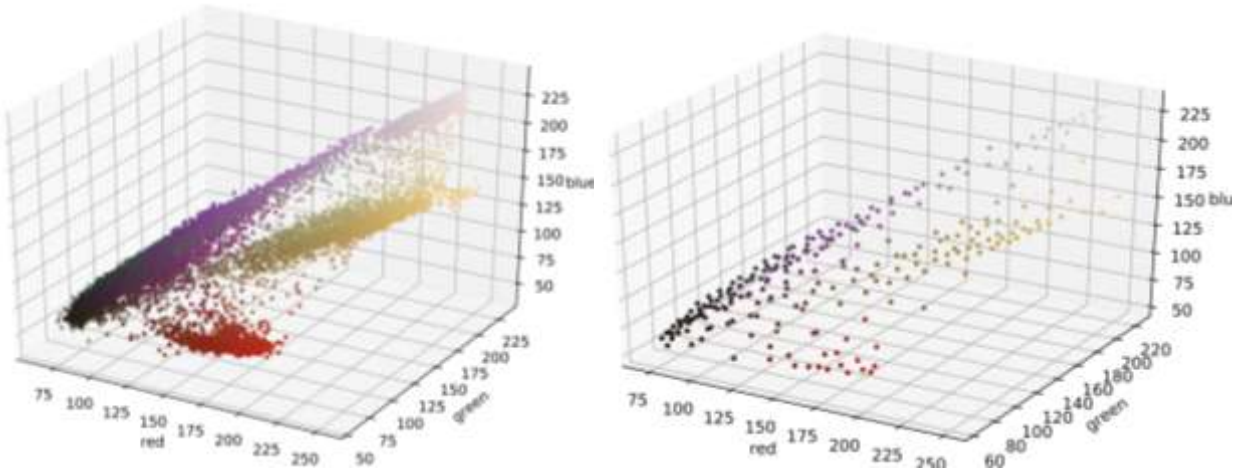
The results of the K-means algorithm are displayed below:

Original image: 8458 colors



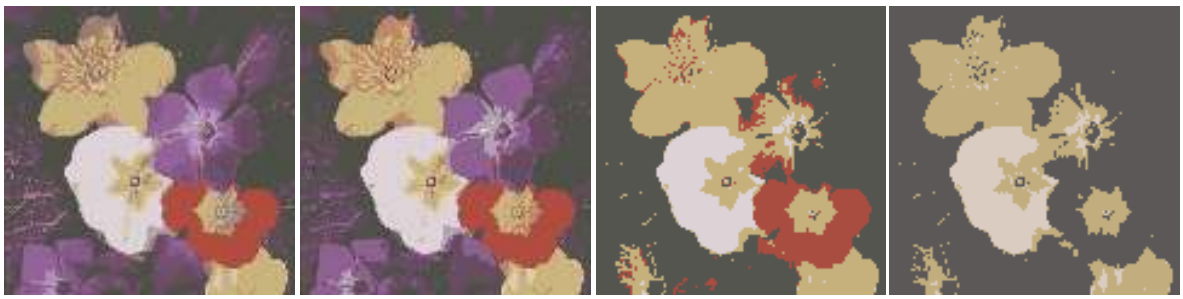
From left to right: k=256, 128, 64, 32, 16, 8, 4, and 2

3d plot of flowersm.ppm in RGB colorspace:

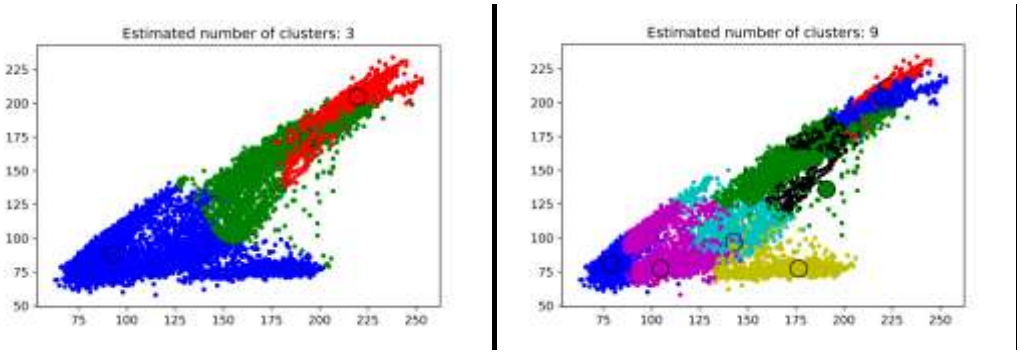


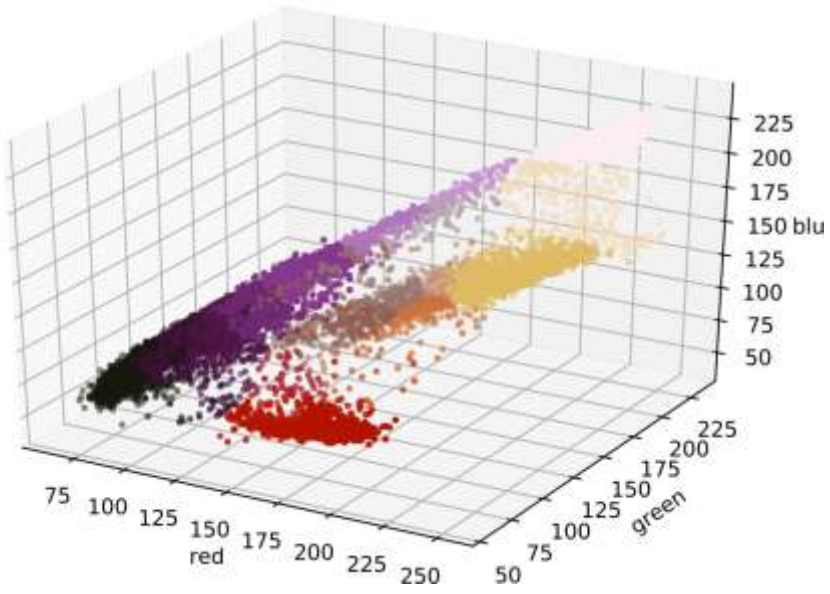
Means Shift

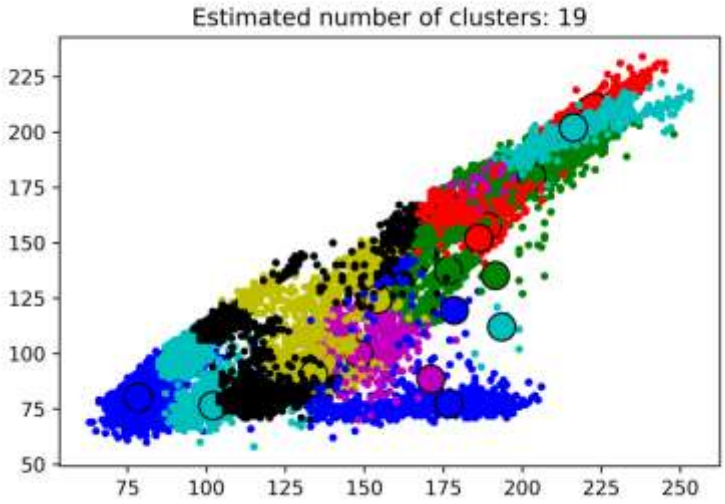
total clusters: 69



From left to right: 19 ,9 ,5 and 3







Classifier performance

There are several methods used to determine the performance of image compression. One means of measuring the accuracy of image compression is by determining the mean squared error, which is simply the mean Euclidean distance between a pixel in the original image and the corresponding pixel in the resultant image.

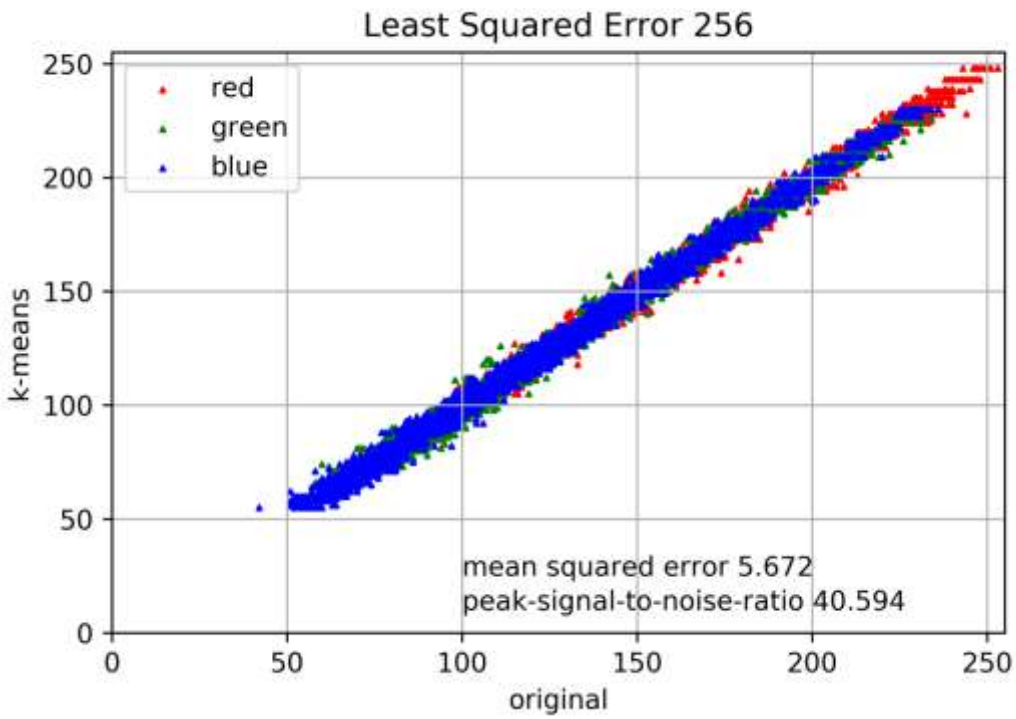
Mean Squared Error:

$$MSE = \frac{1}{mn}\sum_{0}^{m-1}\sum_{0}^{n-1}\|f(i,j) - g(i,j)\|^2$$

Another way of measuring image accuracy is using the peak-signal-to-noise-ratio (PSNR) (Woods and Gonzalez, 1992).

$$PSNR = 20\log_{10}\left(\frac{MAX_f}{\sqrt{MSE}}\right)$$

The PSNR is measured in units of decibels (dB) where higher values indicate more accuracy. The average PSNR for lossy image compression ranges from 30 to 50 dB (Welstead, 1999)



mean squared error	PSNR (dB)	k= (colors)	run time (s)
5.671	40.594	265	13.62
9.088	38.546	128	7.27
14.744	36.445	64	4.17
22.625	34.585	32	2.52
38.439	32.283	16	1.31
60.489	30.314	8	0.24
81.243	29.033	4	0.16
97.401	28.245	2	0.06

References

Comaniciu, D., and Meer, P., 2002, Mean shift: a robust approach toward feature space analysis: IEEE Transactions on Pattern Analysis and Machine Intelligence, v. 24, p. 603–619, doi: 10.1109/34.1000236.

Jones, E., Oliphant, T., and Peterson, P., 2014, {SciPy}: open source scientific tools for {Python}:

K. Fukunaga, and L. Hostetler, 1975, The estimation of the gradient of a density function, with applications in pattern recognition: IEEE Transactions on Information Theory, v. 21, p. 32–40, doi: 10.1109/TIT.1975.1055330.

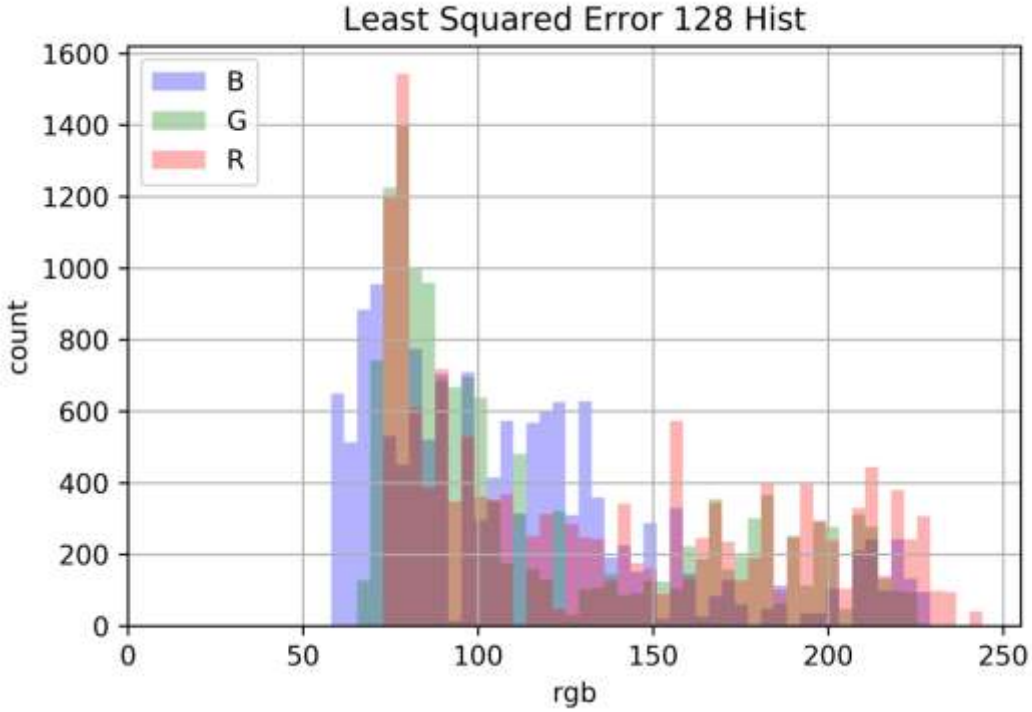
McKinney, W., 2010, Data structures for statistical computing in python, *in* Proceedings of the 9th Python in Science Conference, Austin, TX, v. 445, p. 51–56.

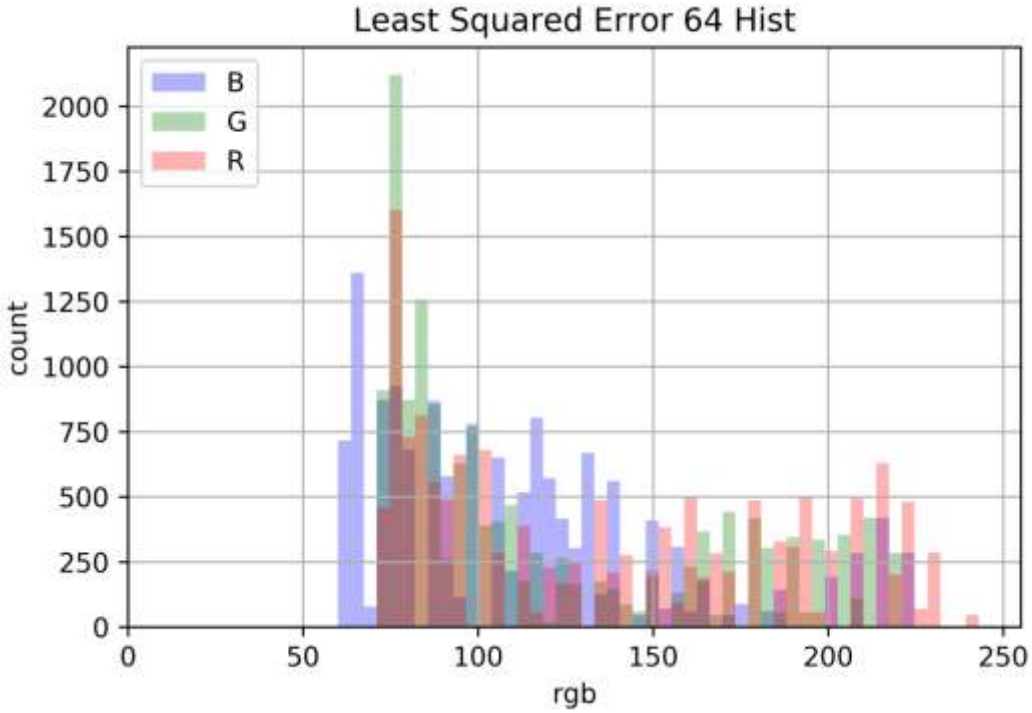
Oliphant, T.E., 2006, A guide to NumPy: Trelgol Publishing USA, v. 1.

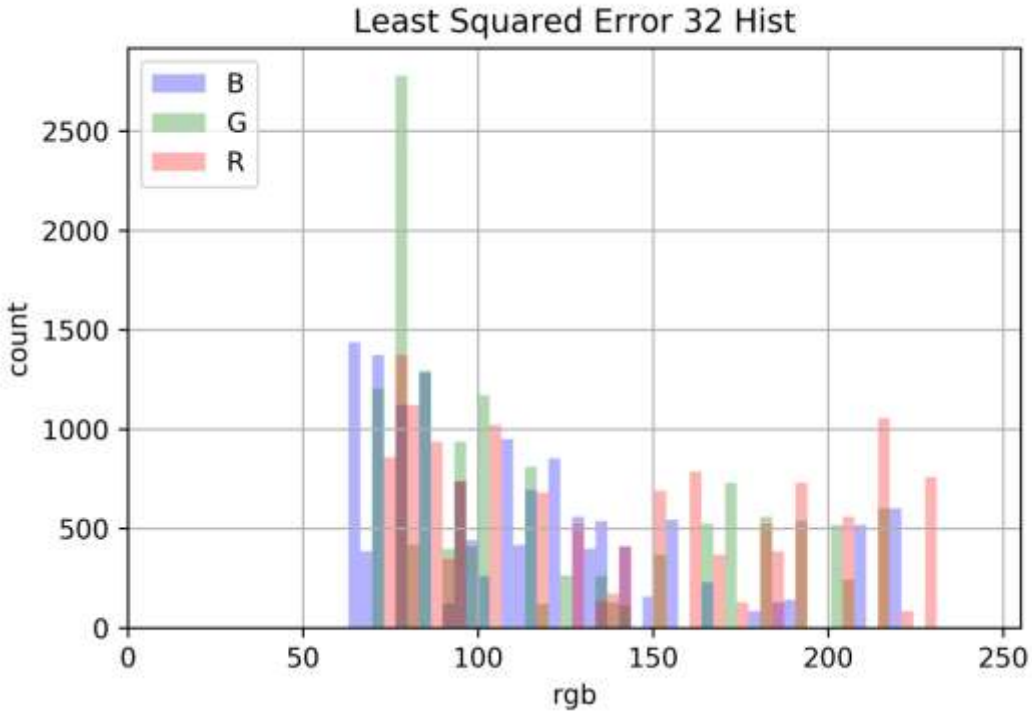
Woods, R.E., and Gonzalez, R.C., 1992, Digital image processing: New York, Addison-Wesley.

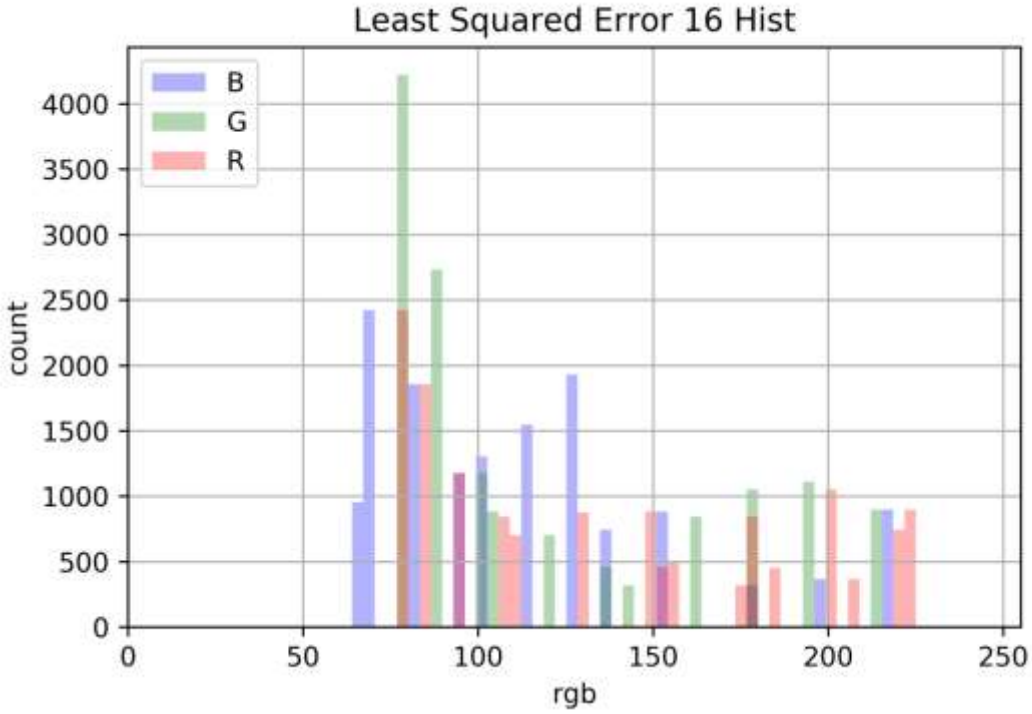
Welstead, S.T., 1999, Fractal and wavelet image compression techniques: SPIE Optical Engineering Press Bellingham, Washington.

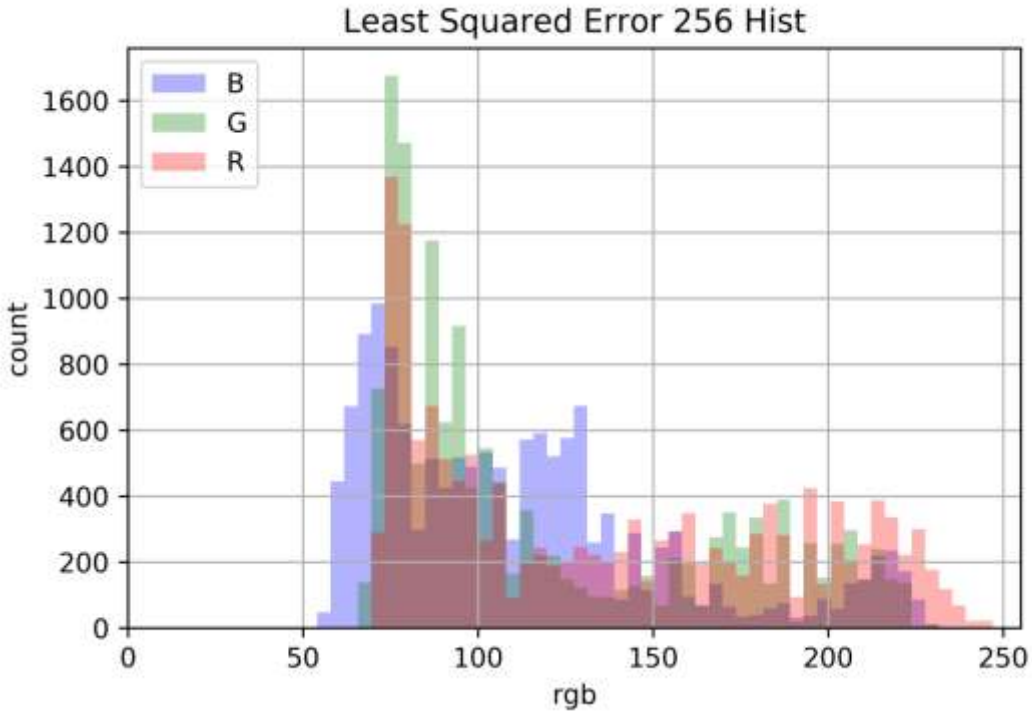
Appendix

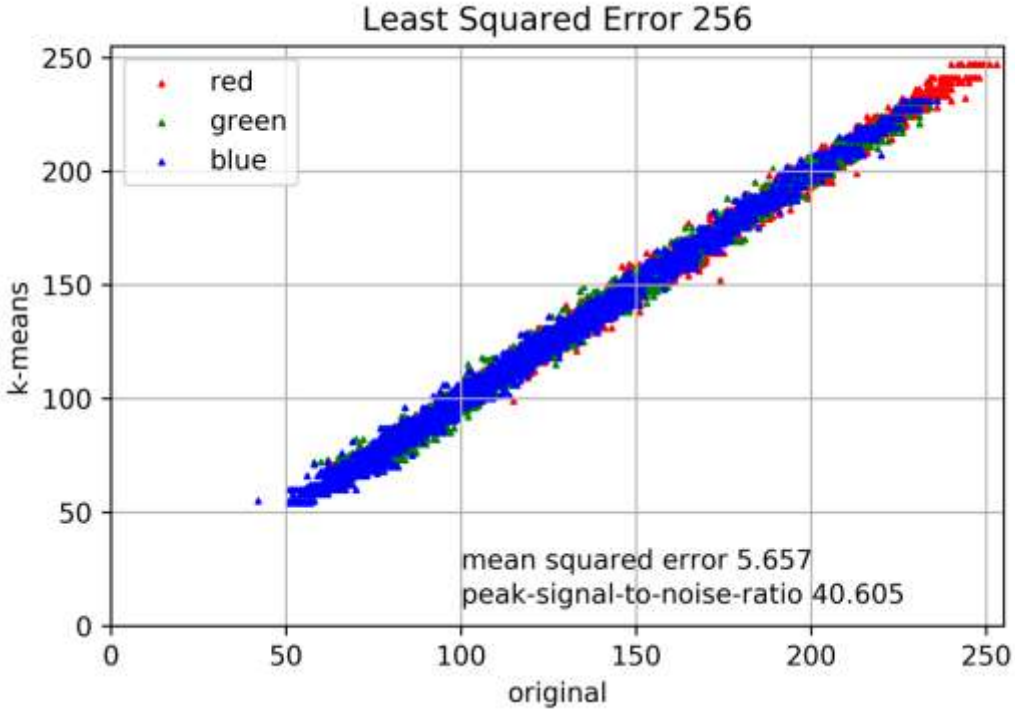

Least Squared Error 128 Hist
B
G
R
count
rgb
0
200
400
600
800
1000
1200
1400
1600
0
50
100
150
200
250

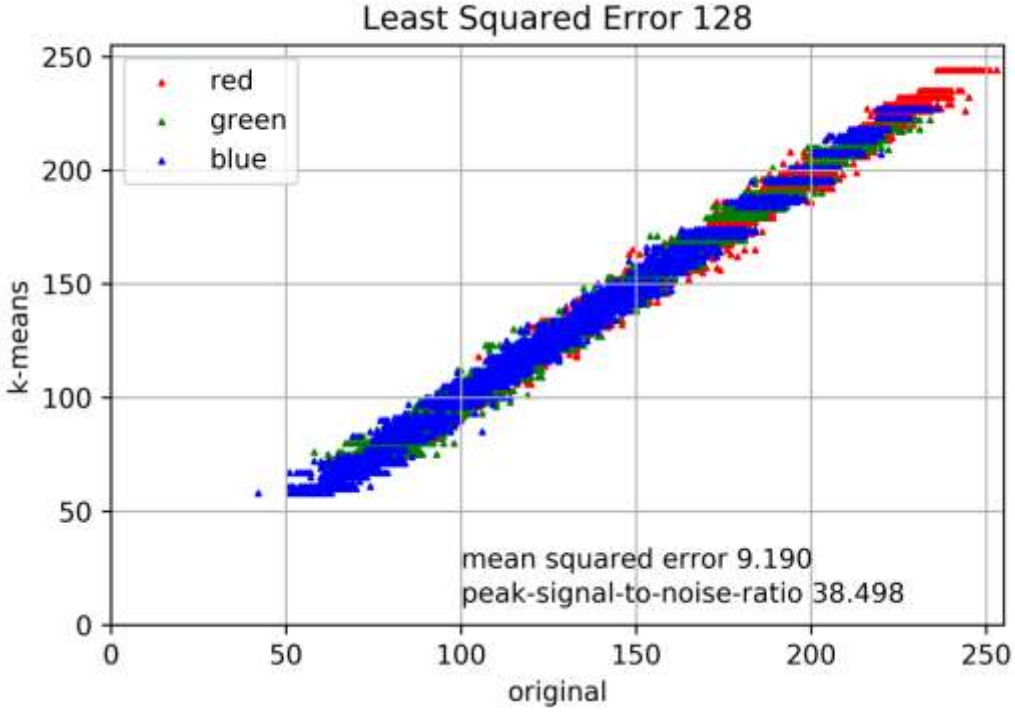

Least Squared Error 64 Hist
B
G
R
count
rgb
0
250
500
750
1000
1250
1500
1750
2000
0
50
100
150
200
250

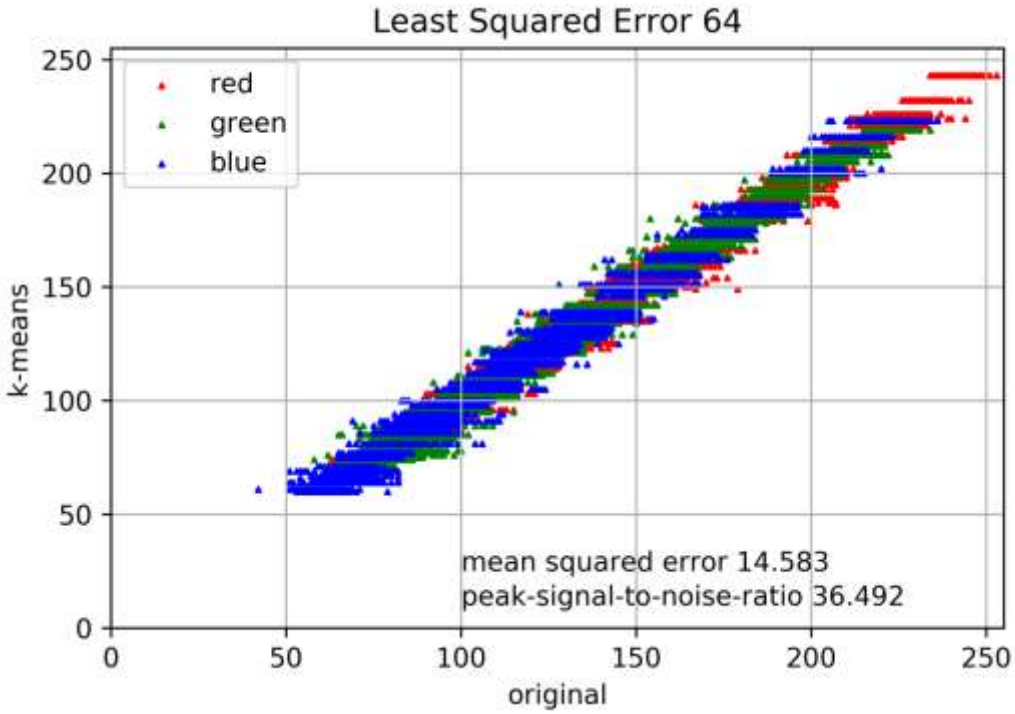

Least Squared Error 32 Hist
B
G
R
count
rgb
0
500
1000
1500
2000
2500
0
50
100
150
200
250

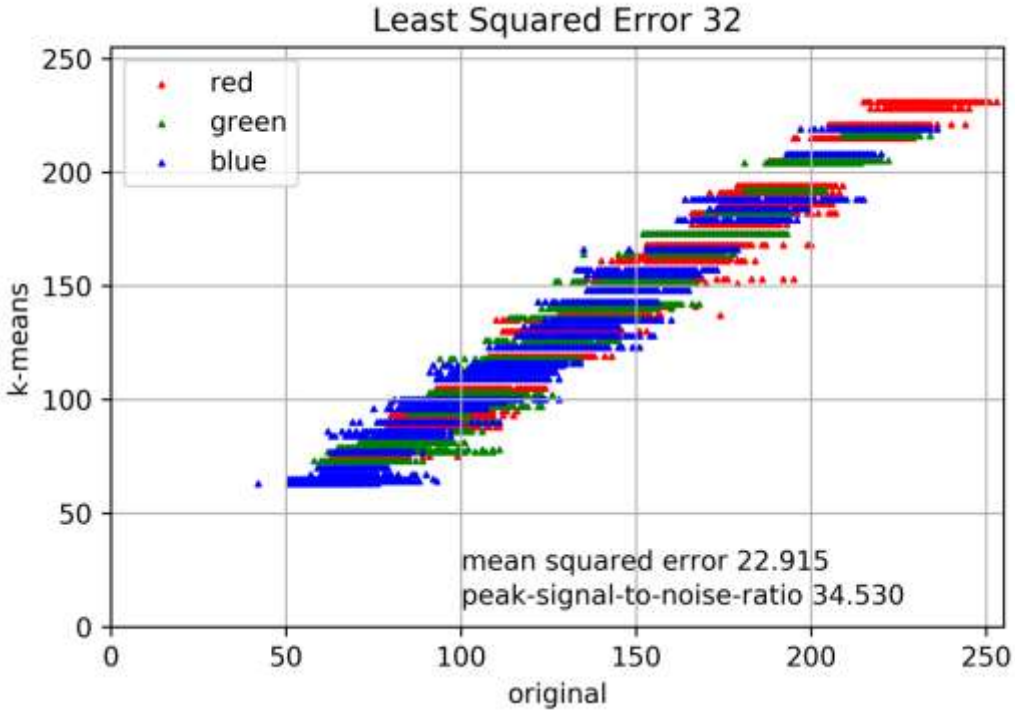

Least Squared Error 16 Hist
B
G
R
count
rgb
0
500
1000
1500
2000
2500
3000
3500
4000
0
50
100
150
200
250

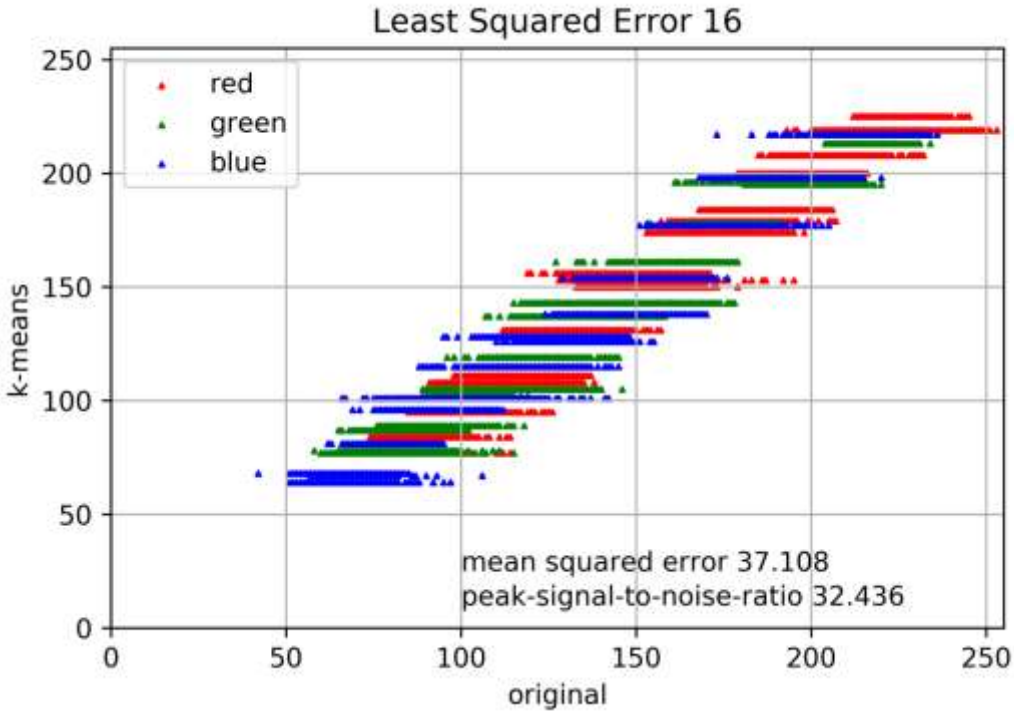

Least Squared Error 256 Hist
B
G
R
count
rgb
0
200
400
600
800
1000
1200
1400
1600
0
50
100
150
200
250


Least Squared Error 256
red
green
blue
k-means
original
mean squared error 5.657
peak-signal-to-noise-ratio 40.605
0
50
100
150
200
250


Least Squared Error 128
red
green
blue
k-means
original
mean squared error 9.190
peak-signal-to-noise-ratio 38.498
0
50
100
150
200
250


Least Squared Error 64
red
green
blue
k-means
original
mean squared error 14.583
peak-signal-to-noise-ratio 36.492
0
50
100
150
200
250


Least Squared Error 32
red
green
blue
k-means
original
mean squared error 22.915
peak-signal-to-noise-ratio 34.530
0
50
100
150
200
250



(please refer to the Jupyter Notebook file included with this document for all code and step by step instructions)