1 Project Description

Processing Image Signals are vital part of various highly raising fields like Deep Learning, Computer Vision etc. These must not only provide high performance but the computations must also be done efficiently and quickly. In this project we have tried to compress the given image data (Gray-scaled and Color Image), by using PCA.

2 Theory:

Normally images and videos have a lot of pixels to retain their clarity which significantly increases its size & slows down the performance when it has to process multiple images or videos. Thus, to overcome this we can use dimensionality reduction which is a Unsupervised Machine Learning techinque.

Principal Component Analysis (PCA), is a dimensionality-reduction method which is used to reduce the dimensionality of a dataset by transforming the data to a new basis where the dimensions are non-redundant and have high variance.

PCA is mathematically defined as an orthogonal linear transformation that transforms the data to a new coordinate system such that the greatest variance by some projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on. In other words, we convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components.

To put it in simpler words, PCA tries to explain as much data variation as it possibly can while discarding highly correlated variables.

3 Methodology

- **Importing the data and subtracting the mean**
  First, we start by converting the given image into numpy array and subtract the mean from each of the data dimensions. This produces a data set whose mean is zero.

- **Calculating the covariance matrix**
  The aim of the covariance matrix is usually to see if there is any relationship between the dimensions. The covariance matrix for a 2 dimensional dataset like images can be expressed as

$$C = \begin{pmatrix} cov(x,x) & cov(x,y) \\ cov(y,x) & cov(y,y) \end{pmatrix}$$

  where

$$cov(x,y) = cov(y,x) = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{n-1}$$

$$cov(x,x) = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(x_i-\bar{x})}{n-1} = \frac{\sum_{i=1}^{n}(x_i-\bar{x})^2}{n-1} = var(x)$$

$$cov(y,y) = \frac{\sum_{i=1}^{n}(y_i-\bar{y})(y_i-\bar{y})}{n-1} = \frac{\sum_{i=1}^{n}(y_i-\bar{y})^2}{n-1} = var(y)$$

- **Computing Eigenvectors and Eigenvalues** The eigenvectors and eigenvalues of the covariance matrix is the fundamental core for Principal Component Analysis. The eigenvectors are responsible for determining the directions of the new feature space while the eigenvalues determine the magnitude. To put it simply, the eigenvalues explain the variance of the data along the new feature axes. The first principal component is the eigenvector with the highest eigenvalue.
  The 'eigh' function from numpy can be used to calculate the solution.

- **Components Selection**
  Once eigenvectors are calculated using the covariance matrix, we order them by their eigenvalue from highest to lowest, which tells you the components in the order of significance. We can decide to ignore the components with lesser significance, so the final data set will have less dimensions than the original.

- **Deriving new data image**
  Once the components (eigenvectors) that we decide to keep in the data image are chosen, we can simply take the transpose of the vector & multiply it on the left of original data, transposed.

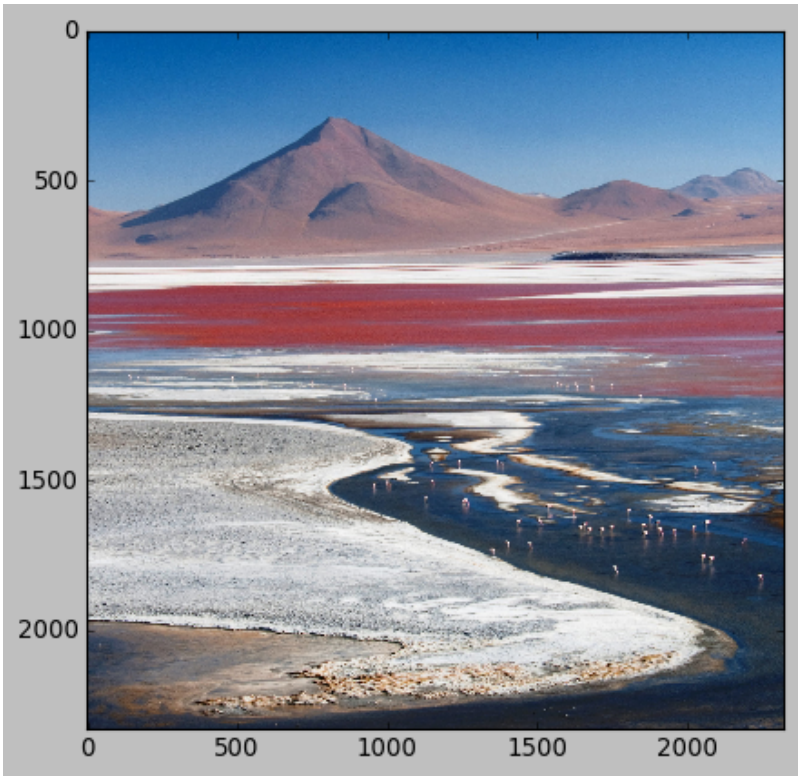
$$NewImage = RowFeatureVector^T * RowDataAdjust^T$$

- **Reconstructing the image back** If we took all the eigenvectors in our transformation will we get exactly the original data back. If we have reduced the number of eigenvectors in the final transformation, then the retrieved data has lost some information.

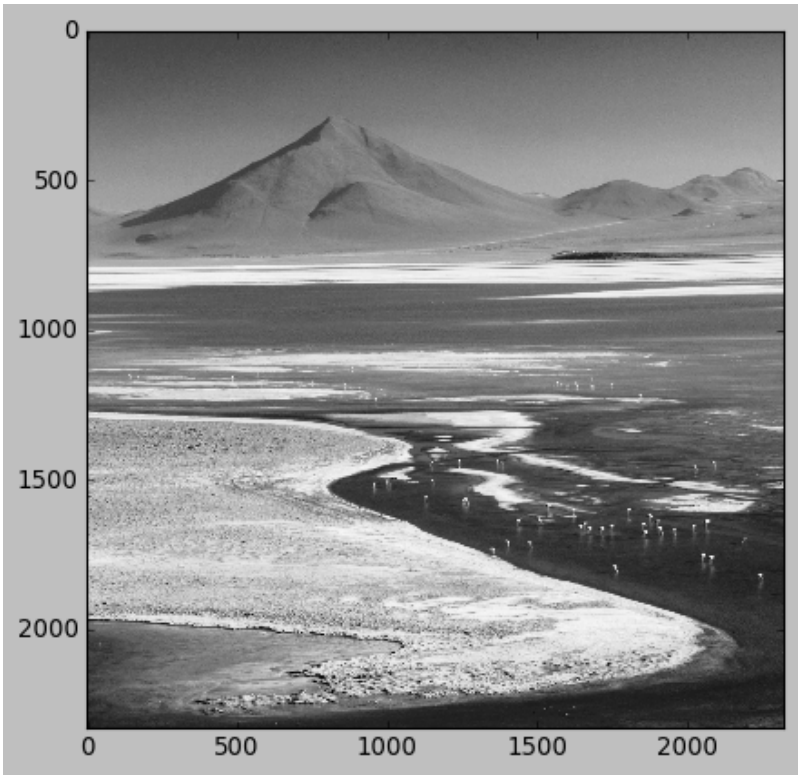
$$OriginalImage = (RowFeatureVector^T * FinalData) + OriginalMean$$

4 Observations

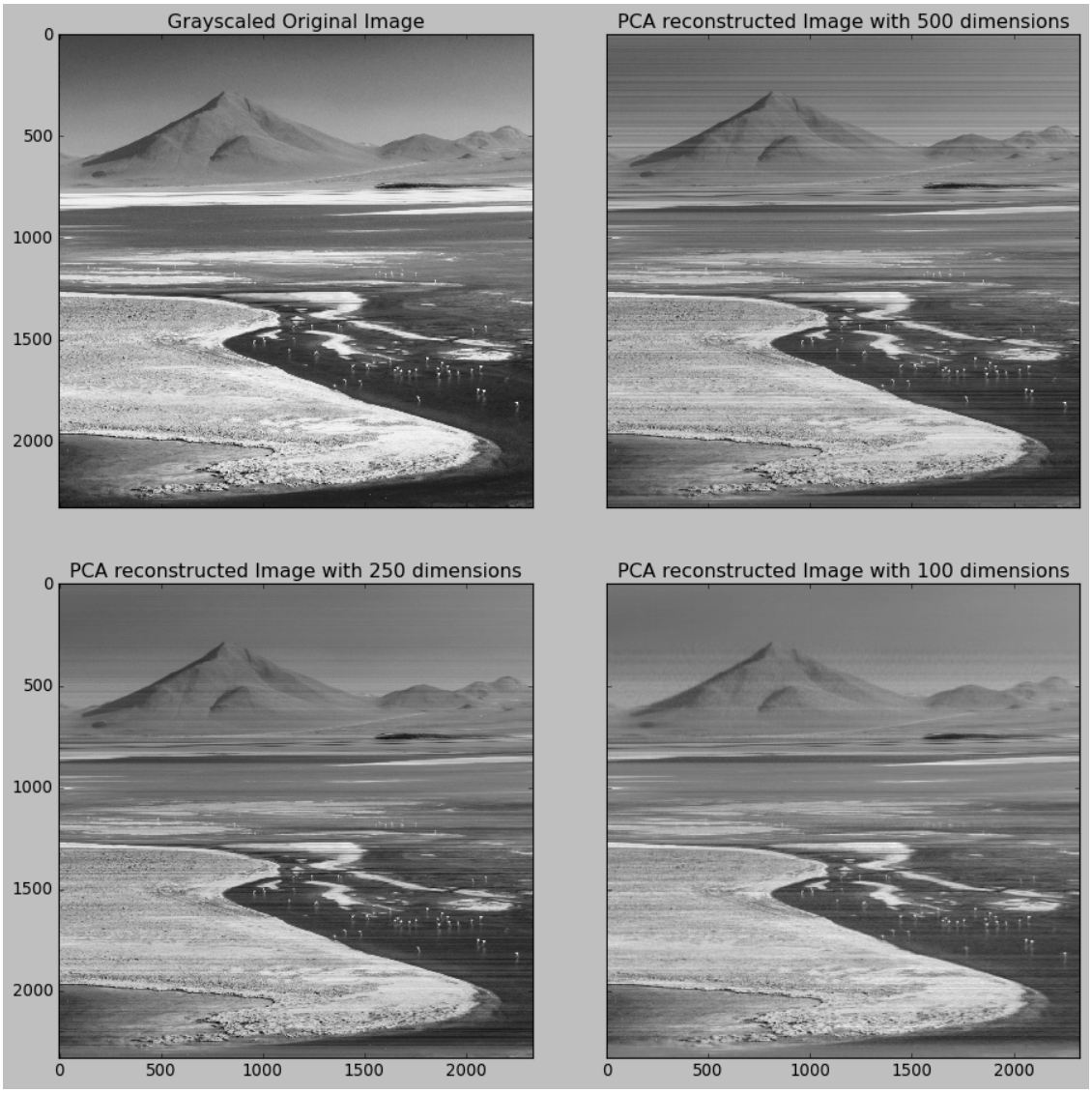
4.1 Original Image



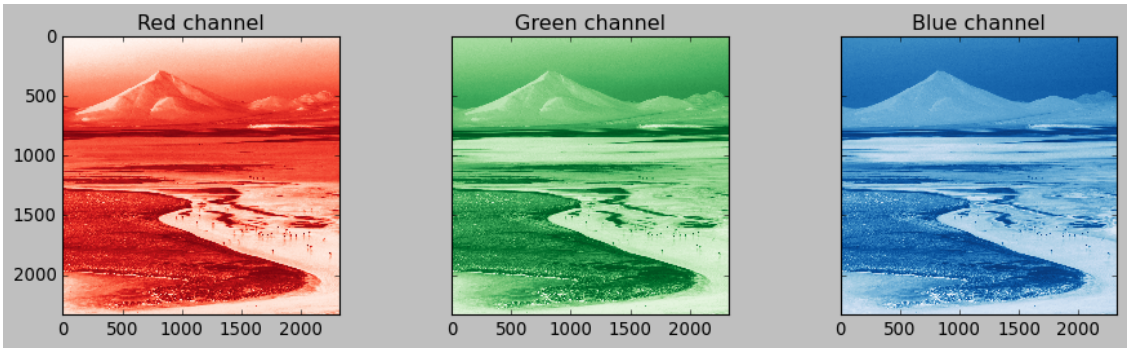
4.2 Gray Scaled Image



4.3 Comparison of Original & PCA reconstructed Images



4.4 RGB Channels of the Image



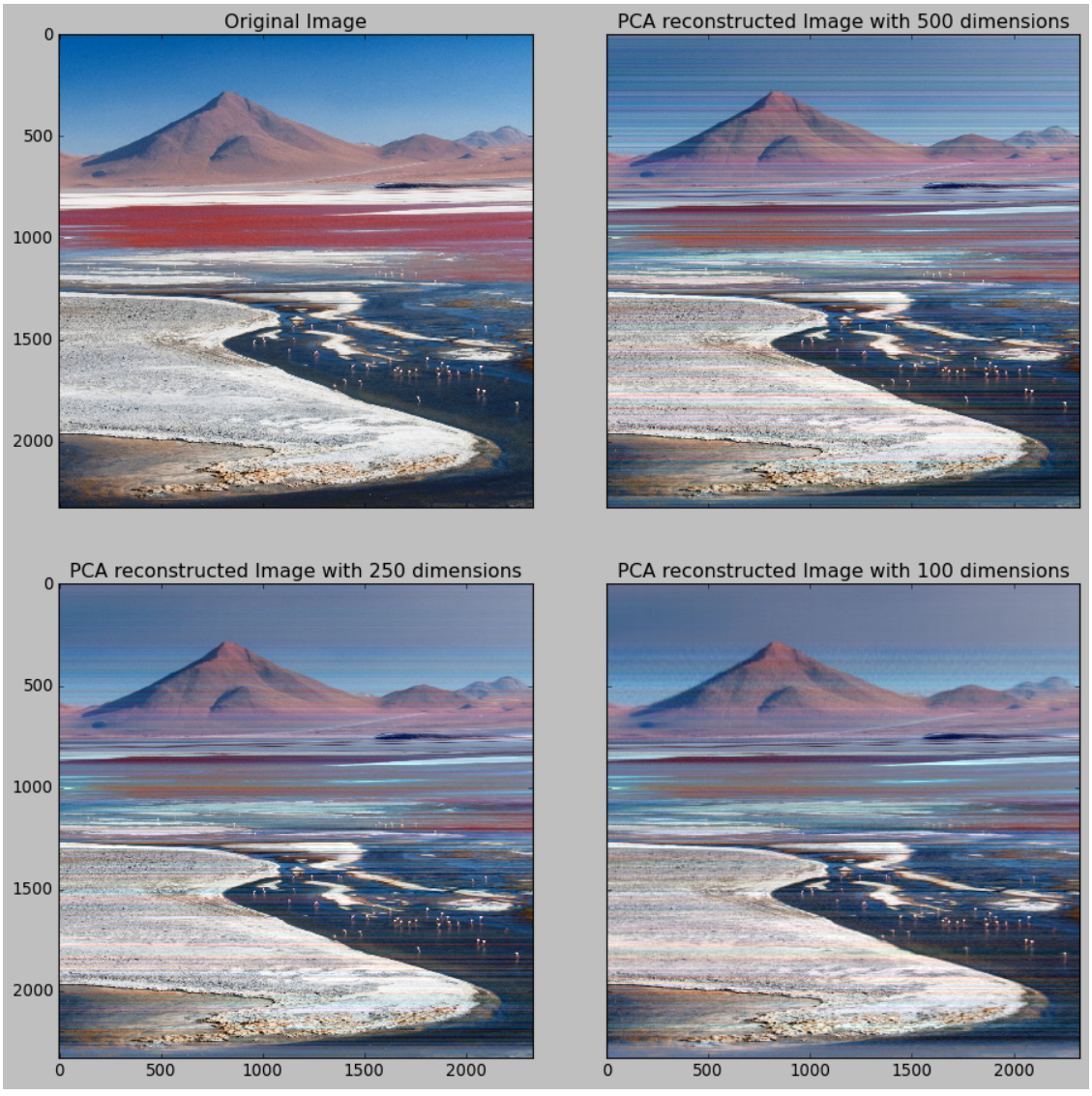
For reducing the dimension of a color image, unlike gray-scaled image, we need to split the RGB channels and then apply the PCA algorithm. All the steps described in the methodology are applied to the red, green and blue channels separately.

We can also perform the same without splitting the RGB channels but it is found that the variance ratio of the same PCA component is better when the channels are splitted.

One the PCA dimensionality reduction is applied to all the 3 channels (R,G,B), we take inverse transform of the 3 channels and then merge the data of all the 3 channels into one.

This allows us to get back the original shape of the image but now the image takes up less space i.e. compressed because it doesn't have unnecessary details from all the dimensions.

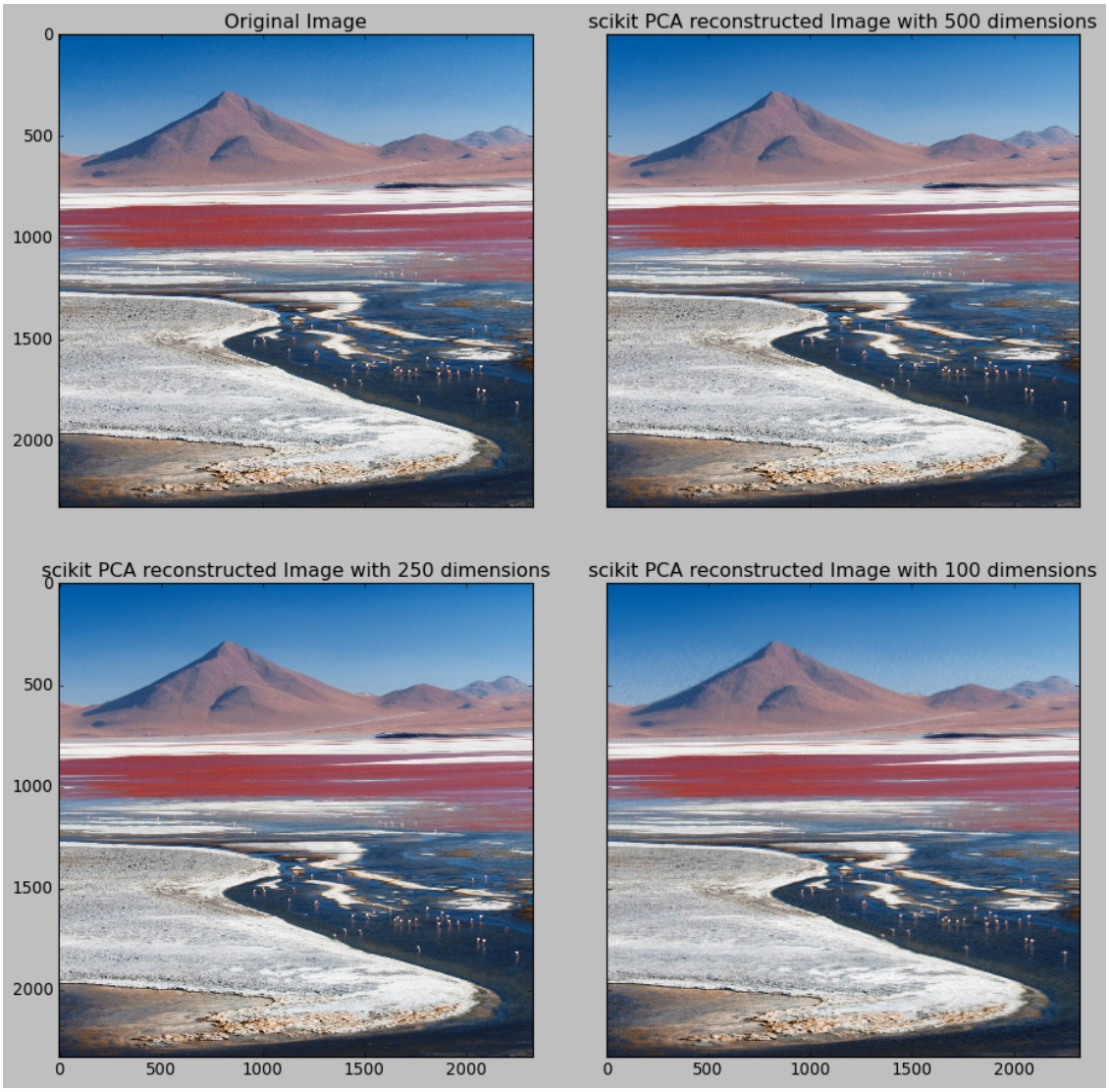
4.5 Comparison of Original & PCA reconstructed Images



4.6 Comparison of Original & sklearn_PCA reconstructed Images

The PCA inbuilt function has been imported from the scikit learn library to compare the performance between the function written mathematically and the sklearn function.

The inbuilt function is more accurate as the library functions are highly optimised with techniques like normalisation to produce the best output. It can be observed in the reconstructed images.

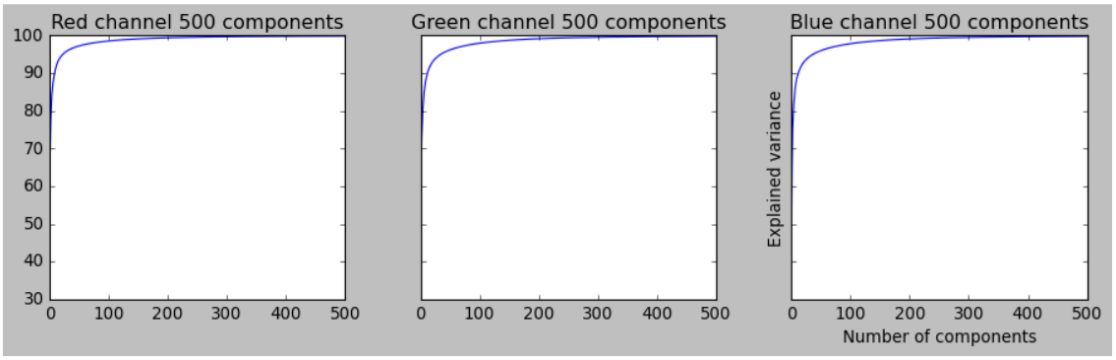


4.7 Explained Variance as function of n_components

The explained variance signifies that just by using 100 components we can keep around 98% of variance in the data. Similarly above 99% of variance for 250 and 500 dimensions.

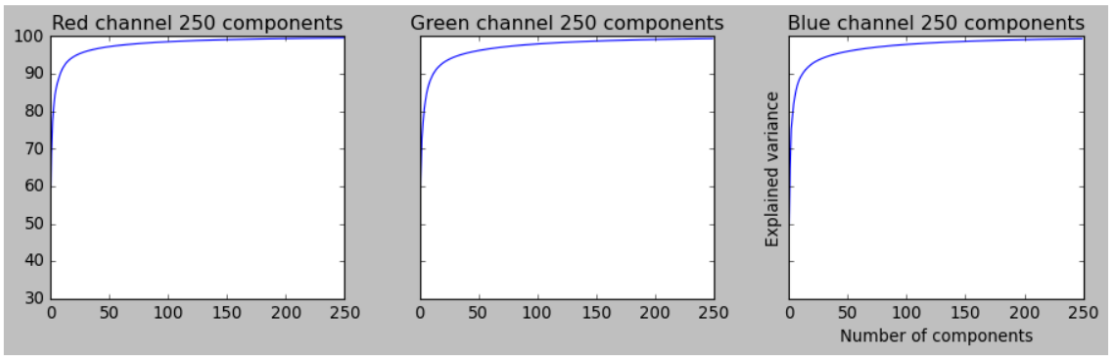
For image with 500 dimensions:

```
(2325, 500)
(2325, 500)
(2325, 500)
Red Channel   : 0.9987085979304723
Green Channel: 0.9981505669323576
Blue Channel : 0.9980114455444808
(2325, 2325) (2325, 2325) (2325, 2325)
(2325, 2325, 3)
```



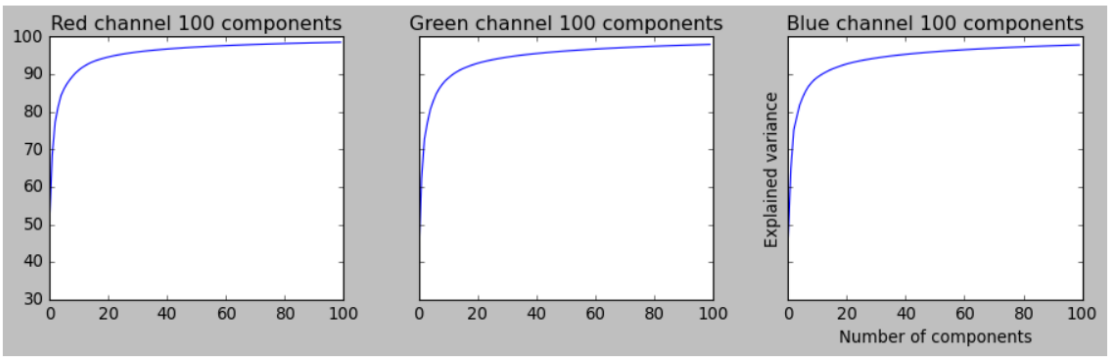
For image with 250 dimensions:

```
(2325, 250)
(2325, 250)
(2325, 250)
Red Channel   : 0.9958410493604414
Green Channel: 0.9940776659032514
Blue Channel : 0.993667175869314
(2325, 2325) (2325, 2325) (2325, 2325)
(2325, 2325, 3)
```



For image with 100 dimensions:

```
(2325, 100)
(2325, 100)
(2325, 100)
Red Channel   : 0.9854984496493708
Green Channel: 0.9795198101165805
Blue Channel : 0.9782465895710076
(2325, 2325) (2325, 2325) (2325, 2325)
(2325, 2325, 3)
```



5 Summary

From the observations, we can that the image is almost reconstructed with just 250-500 components (original image components = 2325). Although there are details missing in comparison to the original image, the reconsturction is pretty close.
Since we don't require high-resolution image data all the time and computations can be done much faster and efficient by reducing some components of the high-resolution image, PCA dimensionality reduction can not only speed up the computation time but can also reduce the storage space of the high-resolution image data.

6 Contributions

- Rithik Kumar
  Understanding the math behind principal component analysis
  Importing the image and converting to grayscale
  Applying PCA for Gray Scale Images

- Kajal
  Understanding the math behind principal component analysis
  Defining pca_reconstruction_color and pca_rgb functions
  Applying PCA for Color Images

- Preethi G
  Understanding the math behind principal component analysis
  Splitting the Color Image into RGB channels
  Applying PCA from sklearn and plotting explained variance
  Organising the final report

7 GitHub - Repository Link

`https://github.com/PreethiGuru/PCA-for-Image-Data-Compression`

8 References

1. `https://en.wikipedia.org/wiki/Principal_component_analysis`
2. `http://www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf`
3. `https://note.nkmk.me/en/python-numpy-image-processing/`