Image Compression using PCA

AIM:

Performing image compression using PCA.

ABOUT:

In PCA, both the compression and the recovery are performed by linear transformations and the method finds the linear transformations for which the differences between the recovered vectors and the original vectors are minimal in the least squared sense.

THEORY:

Principal Component Analysis (PCA)

Let x 1 , . . . , x m be m vectors in R d . We would like to reduce the dimensionality of these vectors using a linear transformation. A matrix W ∈ R n,d , where n < d, induces a mapping x 7→ W x, where W x ∈ R n is the lower dimensionality representation of ax. Then, a second matrix U ∈ R d,n can be used to (approximately) recover each original vector x from its compressed version. That is, for a compressed vector y = W x, where y is in the low dimensional space R n , we can construct $\tilde{x}$ = U y, so that $\tilde{x}$ is the recovered version of x and resides in the original high dimensional space R d .

In PCA, we find the compression matrix W and the recovering matrix U so that the total squared distance between the original and recovered vectors is minimal; namely, we aim at solving the problem.

$$\operatorname*{argmin}_{W\in\mathbb{R}^{n,d},U\in\mathbb{R}^{d,n}} \sum_{i=1}^{m} \|\mathbf{x}_i - UW\mathbf{x}_i\|_2^2.$$

DATASET :

Taken 3 images of 100x100 pixels and taken Blue matrix of its RGB matrices.

PREPROCESSING :

1. Taken 3 images of 100x100 pixels. Converted into numpy array.
2. Reshaped matrices to 1x10000 (100*100) and appended by axis zero to form final X matrix(mxd), here m=3 and d=10,000.
3. Converted each value in the matrix to zero centered value.

4. Covariance the then obtained via np.variance to obtain the variation between the several vectors w.r.t. the other elements in the matrix.

This obtained matrix is the one which will be processed further under PCA (Principal Componeet Analysis)

MODEL DESCRIPTION :

As discussed in the theory, the same model has been followed.

1. We have X vector with the dimensions mxd (3x10000)
2. W matrix with dimensions as mxk(3x2 or 3x1 in our case)
3. U matrix with dimentions as kxm(2x3 or 1x3 respectively in our case)
4. Recontructed matrx obtained as matrix multiplication of UxWxX

Squared Distance between Original and recovered Images:

for k=2

```
sumation=0
for i in range (centered_matrix.shape[0]):
#for j in range (centered_matrix.shape[1]):
    sumation=(X[i]-recon_img_mat[i])**2
np.argmin(sumation)
```

```
0
```

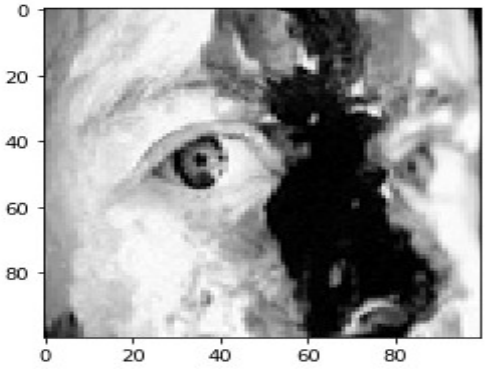
for k=3

```
sumation=0
for i in range (centered_matrix.shape[0]):
#for j in range (centered_matrix.shape[1]):
    sumation=(X[i]-recon_img_mat[i])**2
np.argmin(sumation)
```

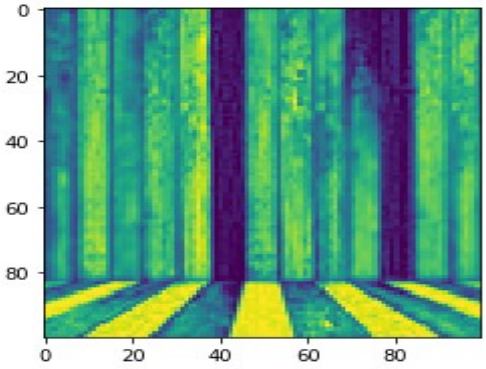
```
32
```

RESULTS:

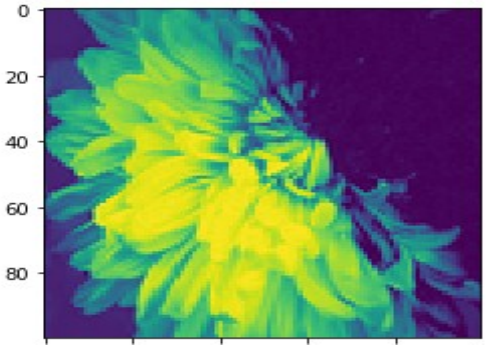
Reconstruction for k=2



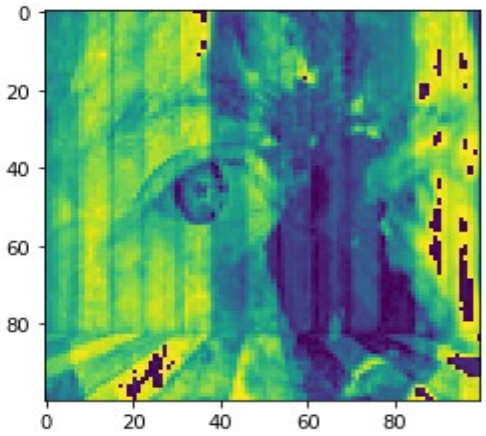

```
In [49]: plt.imshow(recon_img2)
Out[49]: <matplotlib.image.AxesImage at 0x7fc60e3db550>
```




```
In [50]: plt.imshow(recon_img3)
Out[50]: <matplotlib.image.AxesImage at 0x7fc60e3acba8>
```

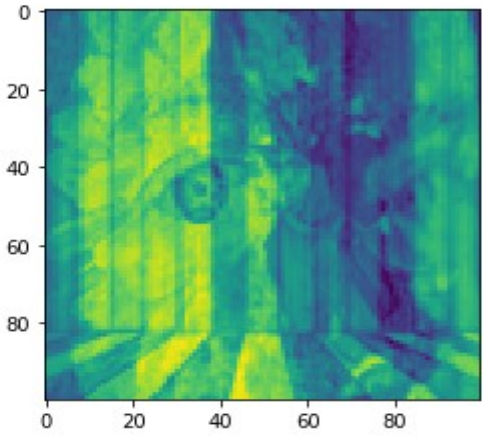


Reconstruction for k=1



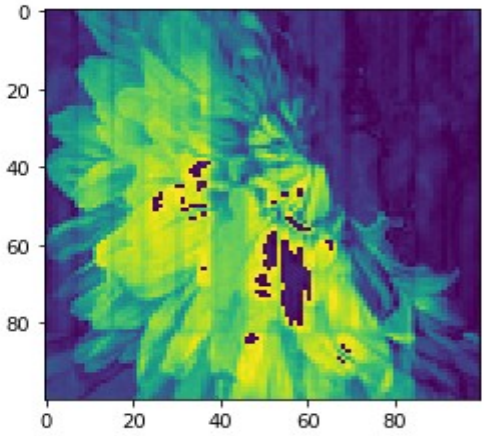

```
In [66]: plt.imshow(recon_img2)
```

```
Out[66]: <matplotlib.image.AxesImage at 0x7fc60dee9518>
```

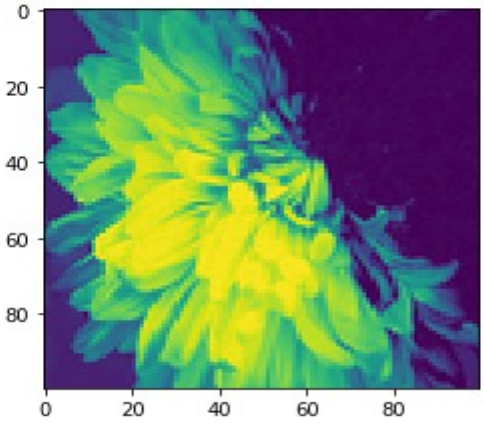



```
In [67]: plt.imshow(recon_img3)
```

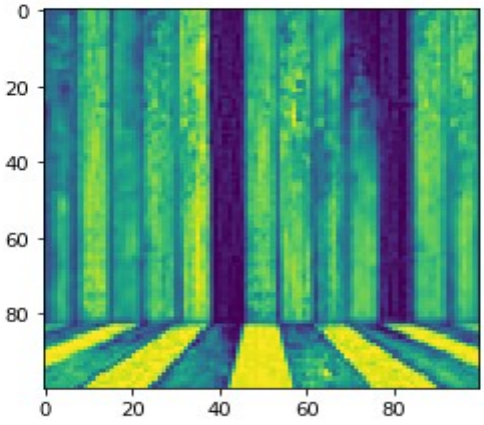
```
Out[67]: <matplotlib.image.AxesImage at 0x7fc60deb9b70>
```



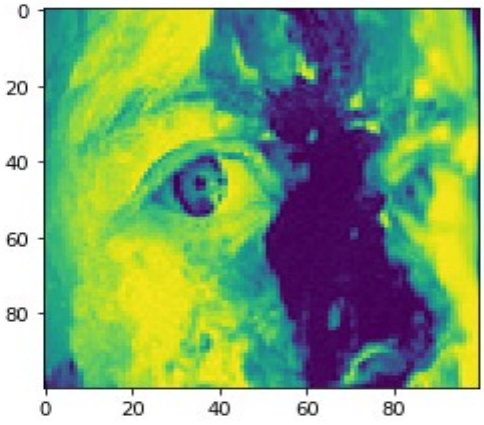
Original Images:




```
In [62]: plt.imshow(img2)
Out[62]: <matplotlib.image.AxesImage at 0x7fc60e022518>
```




```
In [61]: plt.imshow(img1)
Out[61]: <matplotlib.image.AxesImage at 0x7fc60e0c6e10>
```



SETUP:

- Anaconda
- Jupyter Notebook
- Python 3.7
- OS: Linux (UBUNTU 18.04.1)
- Libraries Required: pandas, numpy, matplot library, mpl_toolkits, sklearn.linear_model (Just for the testing predictions)