Application of SVD(Singular value decomposition) for Image compression

* Author: Sachin M Sabariram

* Github: ssr-04

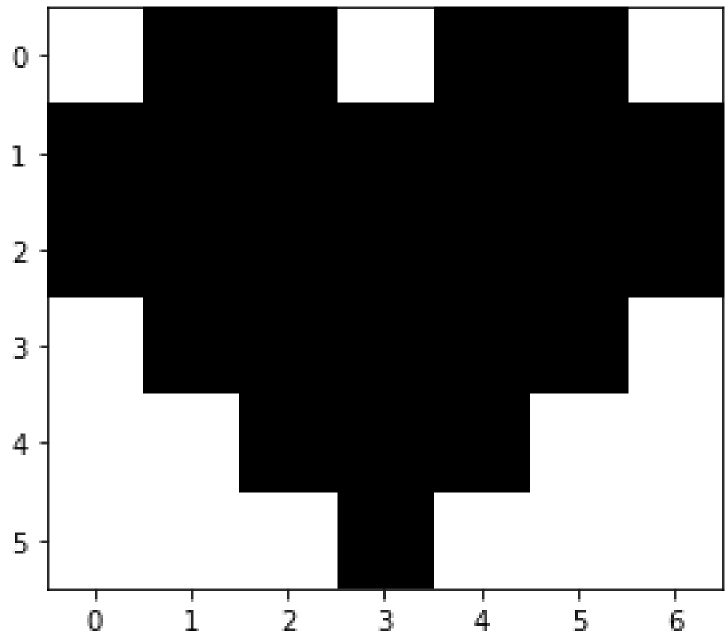
Importing Libraries

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pyplot import imshow
from numpy.linalg import svd   # Singular value decomposition
```

Creating a sample B/W image array

```
# Image array (B/W)

A1 = np.array([[0,1,1,0,1,1,0],
               [1,1,1,1,1,1,1],
               [1,1,1,1,1,1,1],
               [0,1,1,1,1,1,0],
               [0,0,1,1,1,0,0],
               [0,0,0,1,0,0,0],
              ])

# Display in Greyscale
Bias = 1
vmin = 0
vmax= 1
imshow(1-A1, cmap='gray', vmin=0, vmax=1)

# Bias to invert the color, vmin and vmax are scalar norm values (in
case of b/w it's between 0 and 1)
```

```
<matplotlib.image.AxesImage at 0x1170ad700>
```



Prototyping

```
A1.shape # represents rows and cols
```

```
(6, 7)
```

```
len(A1) # represents the rows
```

```
6
```

```
# Getting the SVD of the matrix array
```

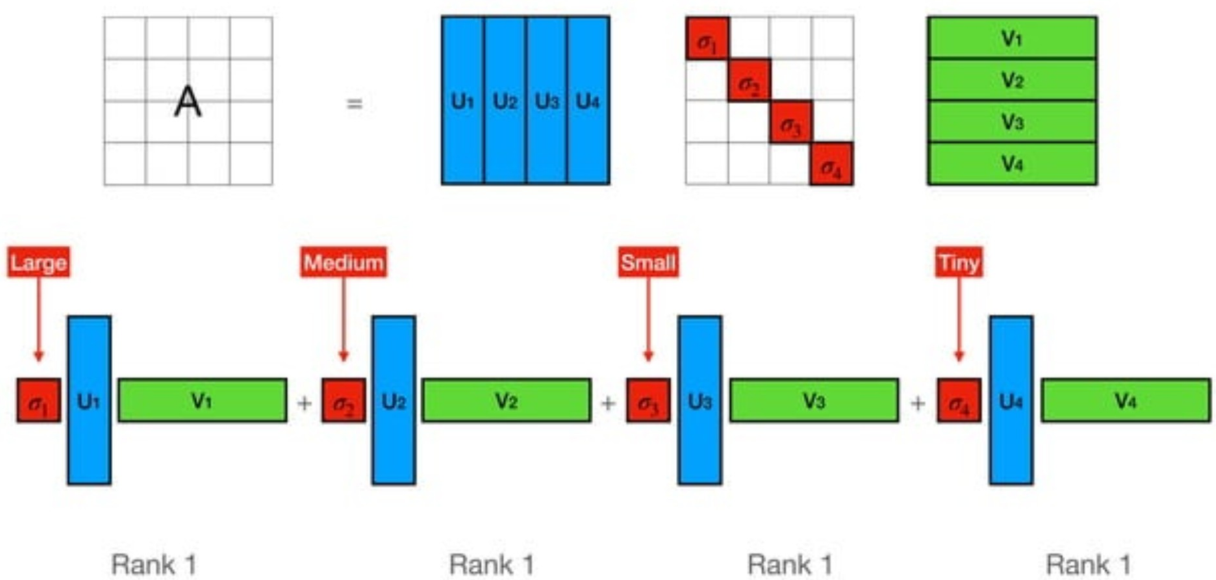
```
U,S,V = svd(A1) # U,V-Orthogonal bases for Transformation, S-Scale of Transformation
```

```
print(np.round(U,2))
print()
sigma = np.diag(S) # Getting the values along the diagonal since, (A=U×S×Vᴛ)
print(np.round(sigma,2))
print()
print(np.round(V,2))
print()
```

```
[[-0.36  0.   -0.73 -0.05 -0.48  0.32]
 [-0.54 -0.35  0.27 -0.08  0.39  0.59]
 [-0.54 -0.35  0.27 -0.08 -0.39 -0.59]
 [-0.45  0.35 -0.27  0.52  0.48 -0.32]
 [-0.28  0.71  0.18 -0.62  0.   -0.  ]
 [-0.08  0.35  0.46  0.57 -0.48  0.32]]
```

```
[[4.74 0.   0.   0.   0.   0.  ]
 [0.   1.41 0.   0.   0.   0.  ]
 [0.   0.   1.41 0.   0.   0.  ]
 [0.   0.   0.   0.73 0.   0.  ]
 [0.   0.   0.   0.   0.   0.  ]
 [0.   0.   0.   0.   0.   0.  ]]

[[-0.23 -0.4  -0.46 -0.4  -0.46 -0.4  -0.23]
 [-0.5  -0.25  0.25  0.5   0.25 -0.25 -0.5 ]
 [ 0.39 -0.32 -0.19  0.65 -0.19 -0.32  0.39]
 [-0.22  0.42 -0.44  0.42 -0.44  0.42 -0.22]
 [-0.42  0.57 -0.03 -0.    0.03 -0.57  0.42]
 [-0.49 -0.38 -0.35  0.    0.35  0.38  0.49]
 [ 0.3   0.18 -0.61 -0.    0.61 -0.18 -0.3 ]]
```

Procedure for Image compression




```python
type(U), type(U) == type(S) == type(V) # Checking datatypes
```

```
(numpy.ndarray, True)
```

Takeaways

- Mat1 = S1 * U1 * V1^T

- U is taken along the column

- S is taken along the diagonal

- V is taken along the row

```
print(f"Shape of U: {U.shape}")
print(f"Shape of S: {S.shape}")
print(f"Shape of V: {V.shape}")
```

```
Shape of U: (6, 6)
Shape of S: (6,)
Shape of V: (7, 7)
```

Note:

- We can see that the size of the matrices vary,

- which makes them not suitable for matrix multiplication so,

- we can just select the size using range (that is equal to the no fo rows of parent matrix)

```
mat = """[[-0.36  0.   -0.73 -0.05 -0.48  0.32]
 [-0.54 -0.35  0.27 -0.08  0.39  0.59]
 [-0.54 -0.35  0.27 -0.08 -0.39 -0.59]
 [-0.45  0.35 -0.27  0.52  0.48 -0.32]
 [-0.28  0.71  0.18 -0.62  0.   -0.  ]
 [-0.08  0.35  0.46  0.57 -0.48  0.32]]

[[4.74 0.   0.   0.   0.   0.  ]
 [0.   1.41 0.   0.   0.   0.  ]
 [0.   0.   1.41 0.   0.   0.  ]
 [0.   0.   0.   0.73 0.   0.  ]
 [0.   0.   0.   0.   0.   0.  ]
 [0.   0.   0.   0.   0.   0.  ]]

[[-0.23 -0.4  -0.46 -0.4  -0.46 -0.4  -0.23]
 [-0.5  -0.25  0.25  0.5   0.25 -0.25 -0.5 ]
 [ 0.39 -0.32 -0.19  0.65 -0.19 -0.32  0.39]
 [-0.22  0.42 -0.44  0.42 -0.44  0.42 -0.22]
 [-0.42  0.57 -0.03 -0.    0.03 -0.57  0.42]
 [-0.49 -0.38 -0.35  0.    0.35  0.38  0.49]
 [ 0.3   0.18 -0.61 -0.    0.61 -0.18 -0.3 ]] """
```

```
# Selecting column along U

for i in range(6): # Length (no of rows of orginal matrix A1)
    print(f"{i} Column = {np.round(U[:,i],2)}")
```

```
0 Column = [-0.36 -0.54 -0.54 -0.45 -0.28 -0.08]
1 Column = [ 0.   -0.35 -0.35  0.35  0.71  0.35]
2 Column = [-0.73  0.27  0.27 -0.27  0.18  0.46]
3 Column = [-0.05 -0.08 -0.08  0.52 -0.62  0.57]
4 Column = [-0.48  0.39 -0.39  0.48  0.   -0.48]
5 Column = [ 0.32  0.59 -0.59 -0.32 -0.    0.32]
```

```
# Selecting rows along V

for i in range(6): # Length (no of rows of orginal matrix A1)
    print(f"{i} Row = {np.round(V[i],2)}")
```

```
0 Row = [-0.23 -0.4  -0.46 -0.4  -0.46 -0.4  -0.23]
1 Row = [-0.5  -0.25  0.25  0.5   0.25 -0.25 -0.5 ]
2 Row = [ 0.39 -0.32 -0.19  0.65 -0.19 -0.32  0.39]
3 Row = [-0.22  0.42 -0.44  0.42 -0.44  0.42 -0.22]
4 Row = [-0.42  0.57 -0.03 -0.    0.03 -0.57  0.42]
5 Row = [-0.49 -0.38 -0.35  0.    0.35  0.38  0.49]
```

```
# Selecting diagnals along S

for i in range(6): # Length (no of rows of orginal matrix A1)
    print(f"{i} Diagonal = {np.round(S[i],2)}")
```

```
0 Diagonal = 4.74
1 Diagonal = 1.41
2 Diagonal = 1.41
3 Diagonal = 0.73
4 Diagonal = 0.0
5 Diagonal = 0.0
```

```
U[:,1].shape, V[1,:].shape
```

```
((6,), (7,))
```

So by multiplying U (6x1) and V (1x7 after reshape) we get matrix of (6x7) which matches the original matrix * Instead od reshape we can use <np.outer> to multiply without reshaping

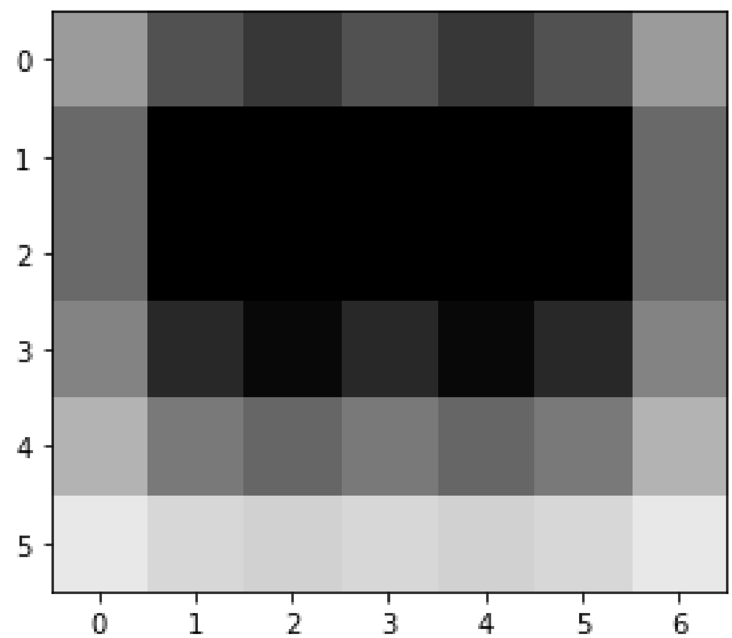
```
sample = np.outer(U[:,0], V[0])
sample.shape
```

```
(6, 7)
```

```
# Finally multiply the U x V with the S to get the Rank 1 matrix with
quality proprtional to value of S
```

```
sample = sample * S[0]
imshow(Bias-sample, cmap='gray', vmin=0, vmax=1)
```

```
<matplotlib.image.AxesImage at 0x1171c8e50>
```

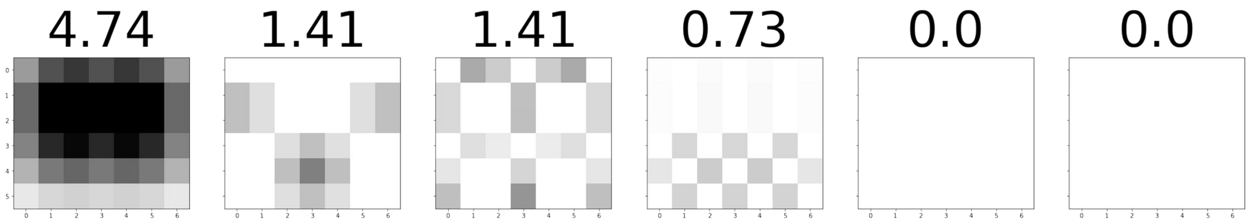


Now we got a image for the 'i' value of 0 (only one feature), ie.with highest priority (due to maximum value of S) Other values of i also contains features in them. * Note all the feature images is of rank 1, we can combine features to get better quality

```
# Creating a array of features

imgs = []
for i in range(6):
    imgs.append(S[i] * np.outer(U[:,i], V[i]))
```

Displaying the images

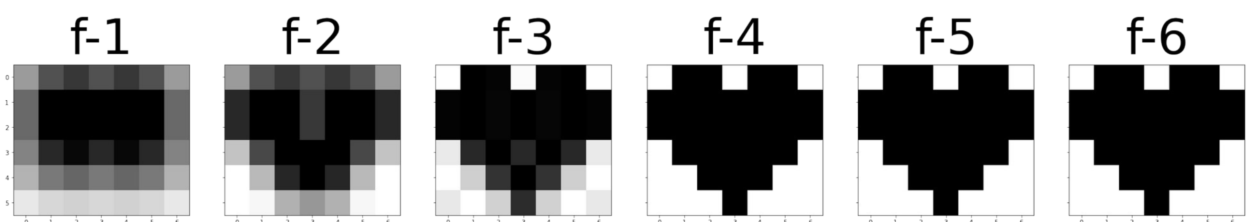
```
# Creating suplots for displaying 6 feature images

fig, axes = plt.subplots(figsize = (6*6,6), nrows=1, ncols=6,
sharex=True, sharey=True)

for n,ax in zip(range(6), axes):
    ax.imshow(Bias-imgs[n], cmap='gray', vmin=0, vmax=1)
    ax.set_title(np.round(S[n],2), fontsize=80)
plt.show()
```



* All of these feature images carry some part of image representation proportional to S value

* We can sum up these features to get a good representation

```
# Combining together the features

combined_imgs = []
for i in range(6):
    combined_imgs.append(sum(imgs[:i+1]))

fig, axes = plt.subplots(figsize = (6*6,6), nrows=1, ncols=6, 
sharex=True, sharey=True)

for n,ax in zip(range(6), axes):
    ax.imshow(Bias-combined_imgs[n], cmap='gray', vmin=0, vmax=1)
    ax.set_title(f'f-{n+1}', fontsize=80)
plt.show()
```

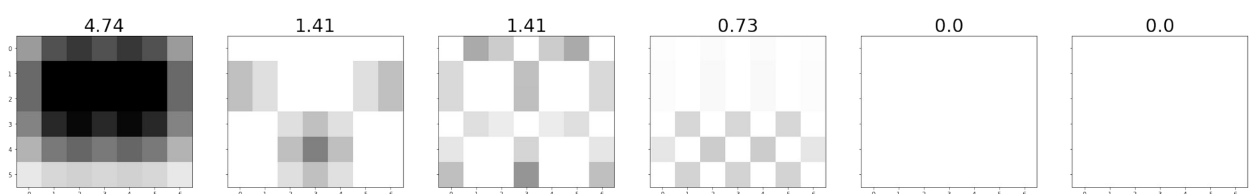


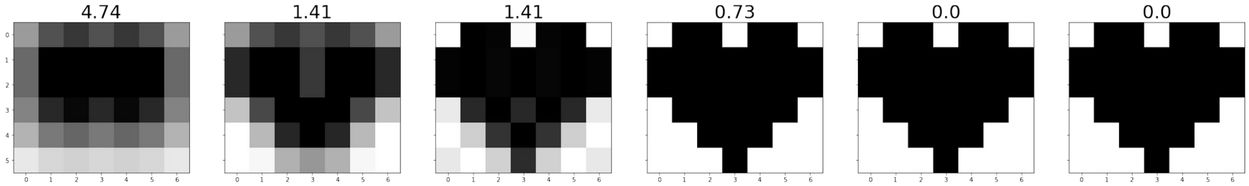
Creating Functions

Plot function

```
def plot_images(img_array, S, n, gray=False, Bias=1, vmin=0, vmax=1, 
sample_no = None):

    """
        This function is used to plot feature images given a image 
array (img_array)

        params:
        * S - represents the S value in SVD (scalig factor)
        * n - represnts no of images
```

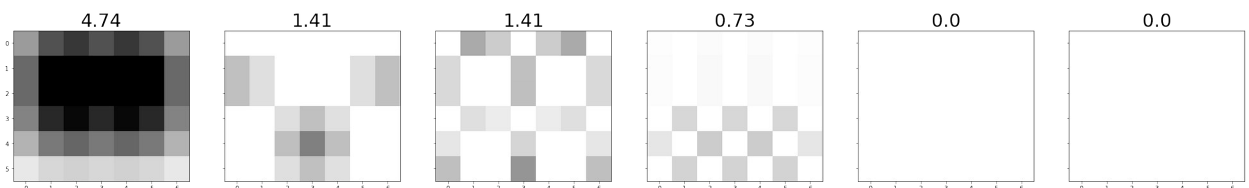
```
        * gray(bool:False) - To display gray/RGB
        * Bias, vmin, vmax are for grayscale adjustment
        * sample_no (array) - represents feature no
    """

    # Creating subplots for n images along the row
    fig, axes = plt.subplots(figsize = (n*n, n), nrows=1, ncols=n,
sharex=True, sharey=True)

    # If sample no is not given (in case of small matrices where we
print all features)
    if sample_no is None:
        for i,ax in zip(range(n), axes):
            if gray:
                ax.imshow(Bias-img_array[i], cmap='gray', vmin=vmin,
vmax=vmax)
            else:
                ax.imshow(img_array[i])
            ax.set_title(np.round(S[i],2), size=30)

    # When sample no is given (in case of actual images) where select
features are displayed as feature size > 50(min)
    else:
        for i,ax in zip(range(n), axes):
            if gray:
                ax.imshow(Bias-img_array[i], cmap='gray', vmin=vmin,
vmax=vmax)
            else:
                ax.imshow(img_array[i])
            ax.set_title(f'{sample_no[i]}({np.round(S[i],2)})',
size=30)

    plt.show()

# Testing plot function for grayscale

plot_images(imgs, S, 6, gray=True) # Individualfeatures
plot_images(combined_imgs, S, 6,gray=True) # Features combined
```

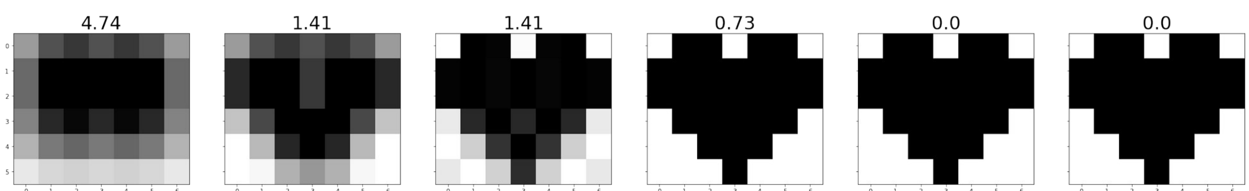




SVD calculation and plotting

```python
def image_compress_svd(img_array):

    n = len(img_array) #gets the no of rows
    U,S,V = svd(img_array) #calculates singular value decomposition

    imgs = []
    for i in range(n):
        imgs.append(S[i] * np.outer(U[:,i], V[i])) #(cols from U and rows from V) * scalar S

    combined = []
    for i in range(n):
        combined.append(sum(imgs[:i+1])) #Feature aggregation upto ith feature

    plot_images(imgs, S, n, gray=True) #Individual features
    plot_images(combined, S, n, gray=True) #Aggregated features

    return U,S,V

d = image_compress_svd(A1) #Test function
```



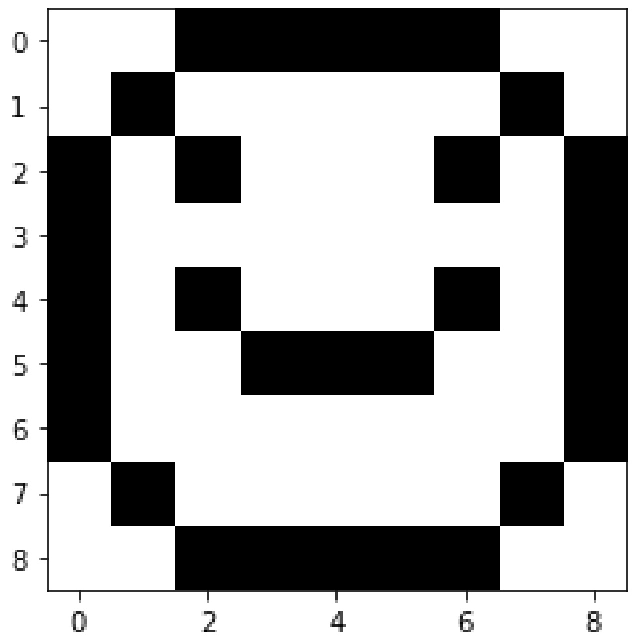


Testing with other shapes

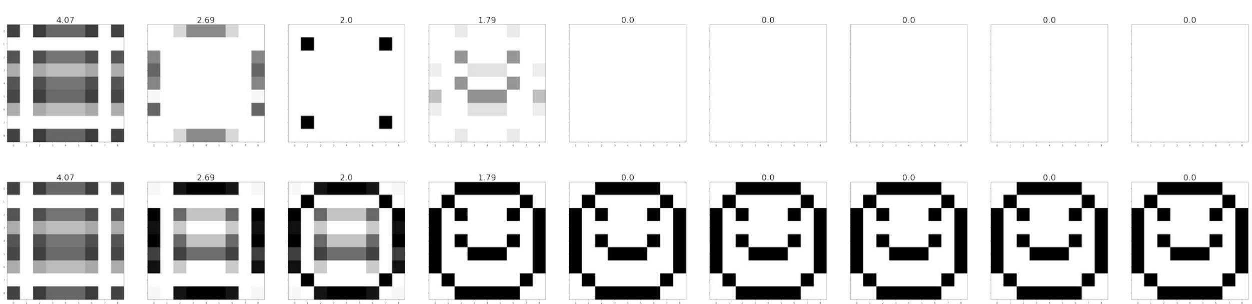
```python
smiley = np.array([
    [0, 0, 1, 1, 1, 1, 1, 0, 0],
    [0, 1, 0, 0, 0, 0, 0, 1, 0],
    [1, 0, 1, 0, 0, 0, 1, 0, 1],
    [1, 0, 0, 0, 0, 0, 0, 0, 1],
```

```
    [1, 0, 1, 0, 0, 0, 1, 0, 1],
    [1, 0, 0, 1, 1, 1, 0, 0, 1],
    [1, 0, 0, 0, 0, 0, 0, 0, 1],
    [0, 1, 0, 0, 0, 0, 0, 1, 0],
    [0, 0, 1, 1, 1, 1, 1, 0, 0]
])
imshow(Bias- smiley, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x117a01fd0>
```




```
d = image_compress_svd(smiley)
```

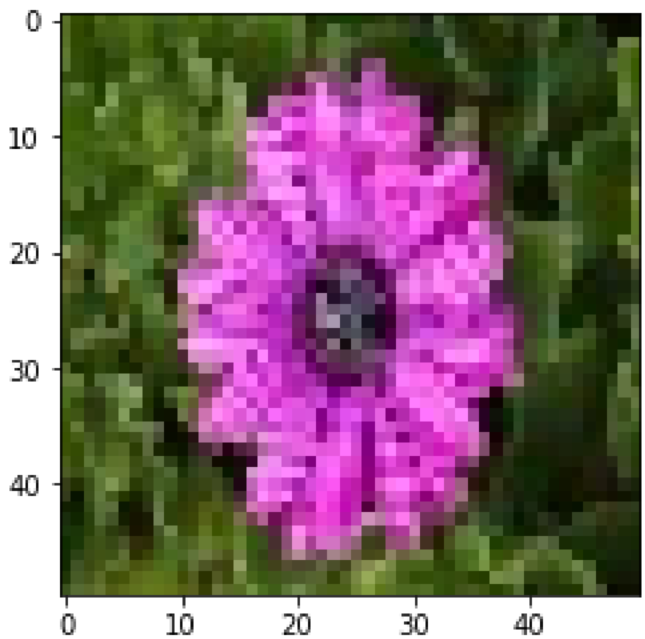


Dealing with RGB Images

```
import cv2
img = cv2.imread("./flower_image.jpg")
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) #By default cv2 reads as
BGR
```

```
imshow(img)
```

```
<matplotlib.image.AxesImage at 0x117c1c9d0>
```

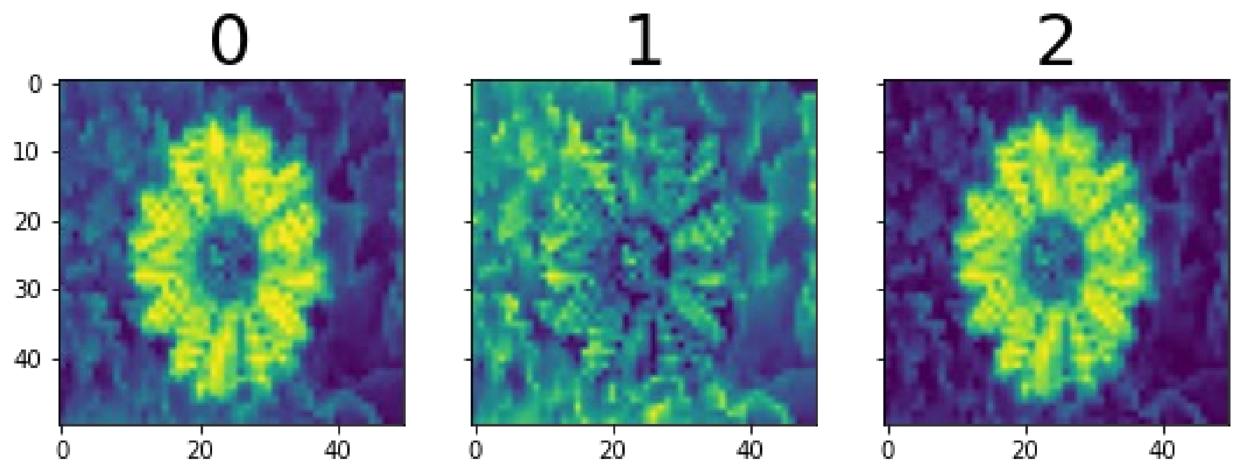


Prototyping

```
img.shape
```

```
(50, 50, 3)
```

```
#(row,col, channel)

r = img[:,:,0] # Red
g = img[:,:,1] # Green
b = img[:,:,2] # Blue
plot_images([r,g,b], [0,1,2], 3)
```

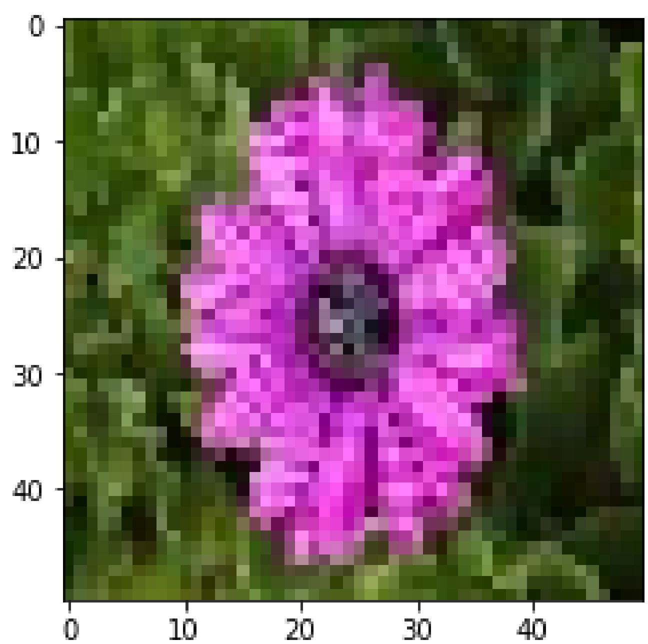



```
comb = np.stack((r,g,b),axis=2) # Combining R,G,B channels together
comb.shape
```

```
(50, 50, 3)
```

```
imshow(comb)
```

```
<matplotlib.image.AxesImage at 0x140873d90>
```



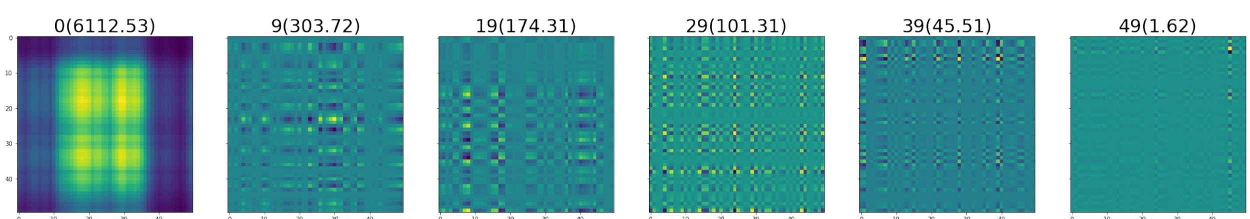
Creating function for RGB

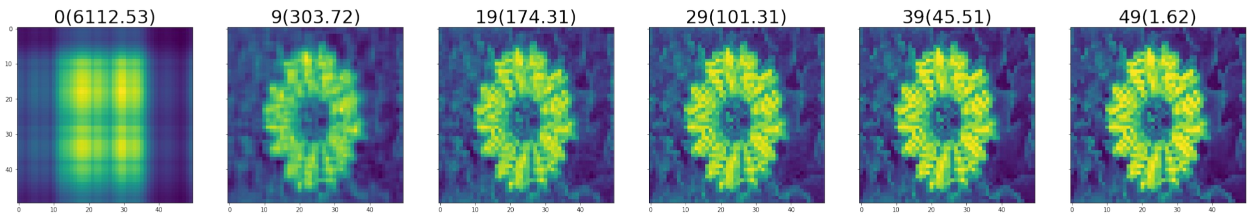
```
def rgb_image_compress_svd(rgb_img_array, n_samples):

    """
        Takes rgb image and no of samples as input and displays the n
```

```
features (equal to n_samples)
        for R,G,B channels individually and those combined.

        n_samples are selected based on equal split from feture 1 to
feature n using linspace.
        (where n is equal to rows of matrix)
    """

    def channel_process(img_array, n_samples=n_samples):
        """
            Processes the SVD for individual RGB channels

            img_array is n x n matrix which represents rows and cols
of each passed channel

            returns combined feature matrix for the channel as well as
feature No
        """
        global samples #defined global so outside function can access

        n = len(img_array) # Gets the no of rows for a channel

        U,S,V = svd(img_array)

        # Individual features
        imgs = []
        for i in range(n):
            if (np.round(S[i],2))!=0: #remove insufficient ones
                imgs.append(S[i] * np.outer(U[:,i], V[i])) # (cols
from U and rows from V) * scalar S


        # Feature Aggregation
        combined = []
        for i in range(n):
            combined.append(sum(imgs[:i+1]))

        # Selecting features to display
        n = len(imgs)
        if n_samples>n: # Error check (no of features should be less
than n)
            n_samples = n

        samples = np.linspace(0,n-1,n_samples) # Equally split between
0 to n_samples to select features
        samples = [int(i) for i in samples] # Convert to Integers

        # Getting only required images
        img_samples = [imgs[i] for i in samples]
        comb_samples = [combined[i] for i in samples]
```

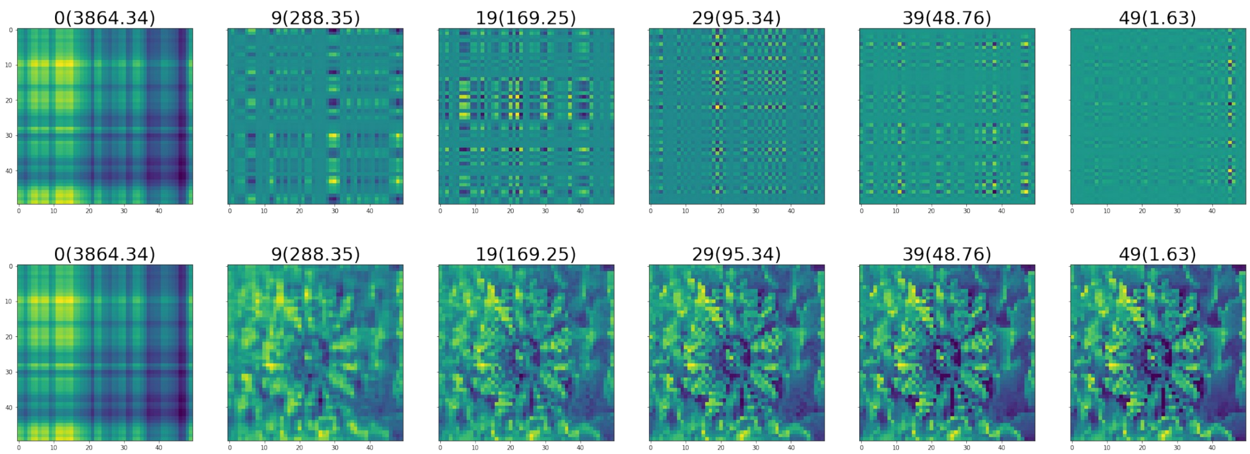
```
        S_samples = [S[i] for i in samples]

        # Displaying images
        plot_images(img_samples, S_samples, n_samples, sample_no =
samples)
        plot_images(comb_samples, S_samples, n_samples, sample_no =
samples)

        return comb_samples, S_samples

    print("Red Channel:")
    r = rgb_img_array[:,:,0]
    c1, s1 = channel_process(r)

    print("Green Channel:")
    g = rgb_img_array[:,:,1]
    c2, s2 = channel_process(g)

    print("Blue channel:")
    b = rgb_img_array[:,:,2]
    c3, s3 = channel_process(b)

    combined = []
    # Combining features of R,G,B together
    for i,j,k in zip(c1,c2,c3):
        combined.append(np.stack((i,j,k), axis=2).astype('uint8'))
    S = []

    # Aggregated scalar value for RGB image
    for i,j,k in zip(s1,s2,s3):
        S.append(np.round((i+j+k)/3, 2))

    print("RGB Combined")
    plot_images(combined, S, len(S), sample_no = samples)

    return combined


comb1 = rgb_image_compress_svd(img, 6)
```

```
Red Channel:
```

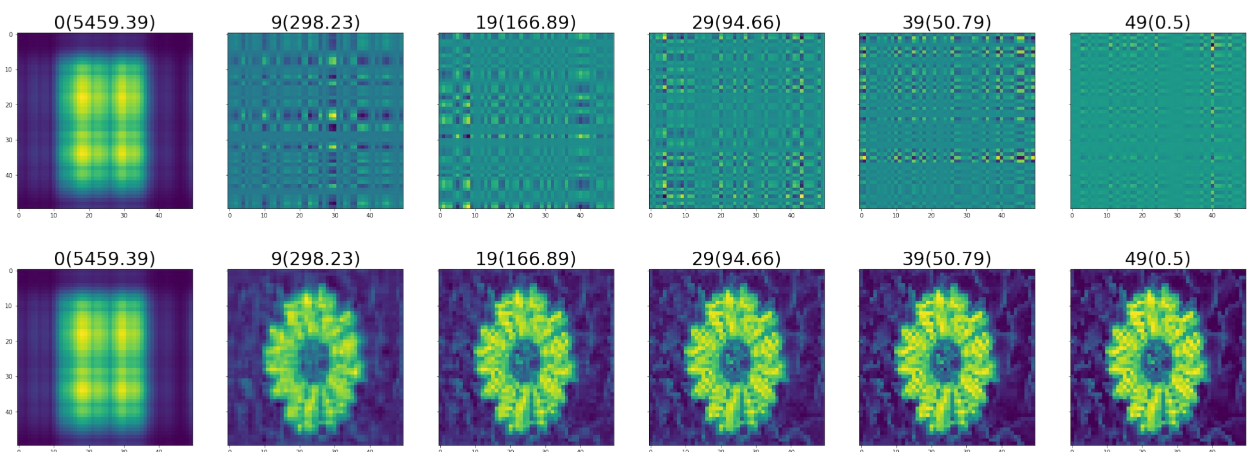




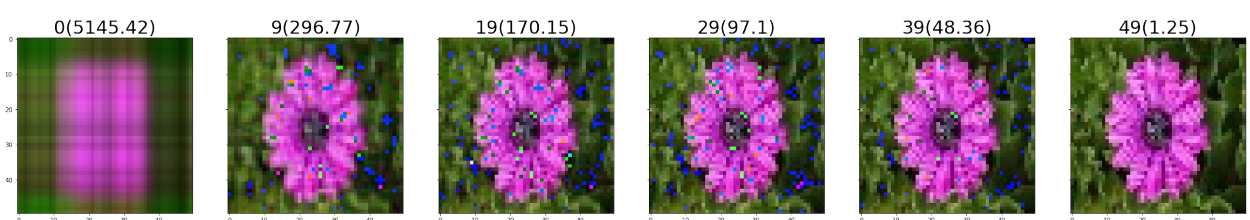
Green Channel:



Blue channel:



RGB Combined



Additional

* How np.stack works

```
p = np.array([[2,3,4],
              [5,6,7],
              [8,9,10]
             ])
q = np.array([[1,2,3],
              [4,5,6],
              [7,8,9]
             ])
r = np.array([[12,13,14],
              [15,16,17],
              [18,19,20]
             ])

p.shape, q.shape, r.shape
```

```
((3, 3), (3, 3), (3, 3))
```

```
(np.stack((p,q,r), axis=-1)) #plugs each feature along the col
```

```
array([[[ 2,  1, 12],
        [ 3,  2, 13],
        [ 4,  3, 14]],

       [[ 5,  4, 15],
        [ 6,  5, 16],
        [ 7,  6, 17]],

       [[ 8,  7, 18],
        [ 9,  8, 19],
        [10,  9, 20]]])
```

Summary:

* Directly applying SVD to an image matrix and reconstructing it won't reduce the storage size in terms of the matrix dimensions. * The compressed image resulting from SVD will have the same dimensions as the original image. * However, the purpose and benefit of using SVD for image compression lie in reducing the amount of information required to represent the image.

* While the file size or matrix dimensions might not change, the essential concept of compression with SVD revolves around retaining only the most significant components of the image. * By discarding less important or lower magnitude components (singular values and corresponding vectors), the reconstructed image represents a simpler version of the original image.

* Primarily used as preprocessing step only to retain the essential information