Singular Value Decomposition: Practical

1807730

October 2022

1 Image Decomposition

One practical example of singular value decomposition is in case of an image. For greyscale images we can take the luminosity value at each pixel and use this to construct a matrix representation of the image. In the case of coloured images, each pixel is represented as a tuple of three integers indicating the intensity of the red, green and blue components. Consider figure 1:

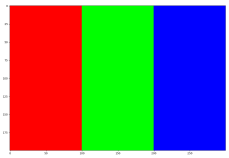

Figure 1: Test Image for SVD

This image is 200×300 in size. It's split into 200×100 segments with each part taking the maximum intensity of its respective colour. The colour of each pixel in the red section is $(255, 0, 0)$, in the green section it's $(0, 255, 0)$ and the blue section is $(0, 0, 255)$. We can break this image down into three images where each pixel is the intensity of the respective colour:

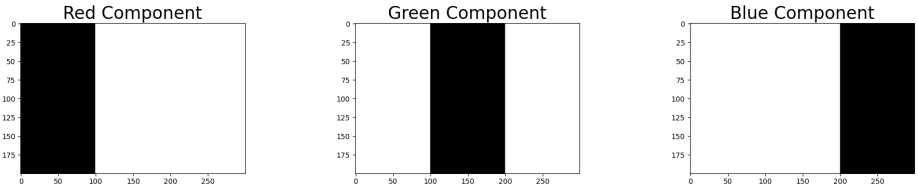

Figure 2: RGB Components of Test Image

Note that each component is now a greyscale image because each pixel is only taking values between 1 and 255 inclusively.

2 Singular Value Decomposition (SVD)

We can represent our original image as three matrices of RGB components. On each of matrix, we can perform Singular Value Decomposition (SVD):

```
# perform the SV decomposition
def __perform_svd(self, matrix: np.matrix) -> tuple:
        return np.linalg.svd(matrix)
```

The code above wraps around a built in function from Pythons NumPy package. This has been done to integrate the functionality into a class object

containing information about the image and a number of other methods. It takes a NumPy matrix object as its input and returns a tuple of three elements containing the $\mathbf{U}\Sigma\mathbf{V}^\top$ decomposition. The second element of the tuple is an ordered array containing the diagonal elements of Σ to easily extract the singular values. We next need a function to retrieve the variance explained by a singular value:

```
# get the variance explained by the singular values
def var_singular(self, matrix: np.matrix, scale: bool = True) ->
↪  numpy.ndarray:
        if scale:
                matrix = self.__scale_matrix(matrix)
        matrix_U, matrix_s, matrix_V = self.__perform_svd(matrix)

        var_explained = np.round(matrix_s ** 2 / np.sum(matrix_s
        ↪  ** 2), decimals = 3)
        return var_explained
```