Principal Component Analysis*

Leonardo Rezende Costa - RA262953**

I. Introduction

This report describes the work done related to the fifth project proposed on class. The aim of this project is to compress an image using the principal component analysis by taking just the **k** first elements on the *singular value decomposition* arrays,

II. Script execution

This section describes how to execute the script and it's dependencies.

The ***t5.py*** file have the script used in this work to generate the results. It can be executed by ***python3 t5.py -i inpyt_path -k k -o output_path*** in an environment with all the dependencies listed on subsection II-A

A. Dependencies

The dependencies to run the python3 scripts are:

- Numpy
- Opencv-python

III. Implementation

This section describes how the PCA based on SVD functions works and what was the approach to develop the script.

A. SVD

As the project explains, our goal is to find the Singular Value Decomposition arrays to calculate a simpler image.

The image 1 illustrates the decomposition of the original array into the matrix $U \sum V$.

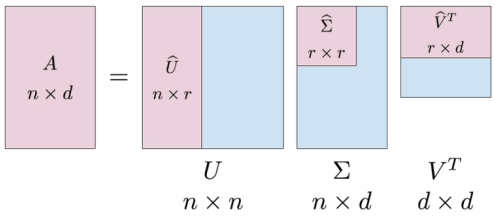

Fig. 1: Singular Value Decomposition illustration. The **r** indicates the number of elements of each matrix. Credits: [2]

The script is simple, as numpy has a function to decompose the matrix and then we just need to multiply choosing k columns for the matrix U, k rows and columns for the diagonal of matrix $\sum$ and the k rows for the transpose of V.

The Listing 1 shows the function that generates the result image using Single Value Decomposition. The $U \sum V$ of each channel is calculated and then used to build the result. Its the core of the project.

```
def svd(img, k):
    # prepare the output array
    A = np.zeros(img.shape)
    # compute each channel separately
    for i in range(img.shape[2]):
        U,s,V = np.linalg.svd(img[:,:,i],
    full_matrices=False)
        A[:,:,i] = np.dot(np.dot(U[:,:k],np.diag(s)
    [:k,:k]), V[:k,:])

    return A
```

Listing 1: Image reconstruction with K components of SVD.

IV. Results

The Fig. 2 shows the images[1] used to test the script.

Fig. 2: Original images used as input on the tests.

Seven tests per input were made to check the script with k = {1,5,10,20,30,40,50}. Fig. 3 shows the result for using just the first value to recompute the image for each input. We can notice that the result shows the average colors of the image, but it doesn't have any details.

*MO443 report - Introdução ao processamento de imagens digitais

**Instituto de Computação - Mestrado, Universidade Estadual de Campinas

[1]All those images was get at <https://www.ic.unicamp.br/~helio/imagens_coloridas/>, accessed in 31/10/2019

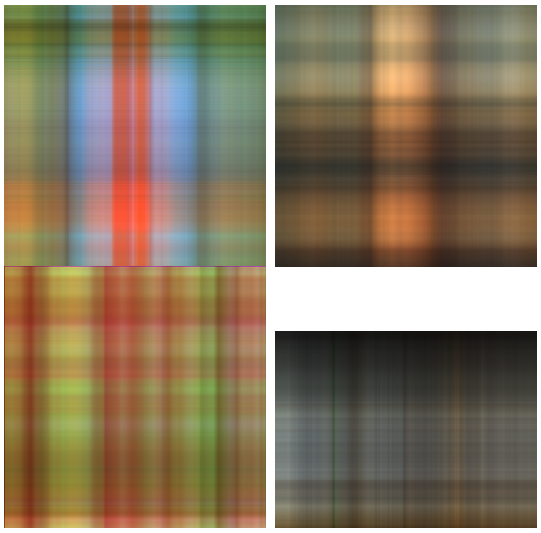

Fig. 3: Results for every input using k = 1. From top-left to bottom-right: baboon, monalisa, peppers and watch.

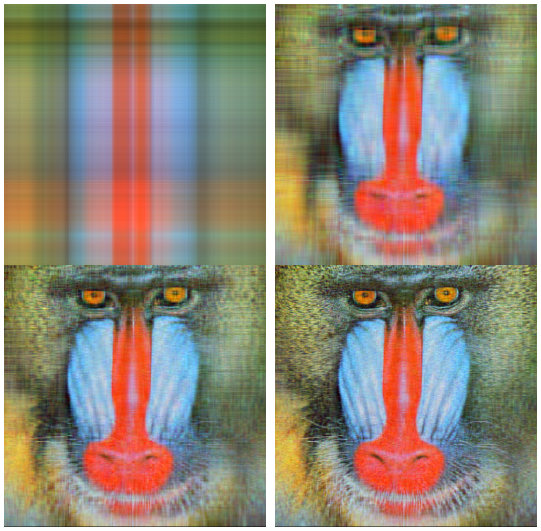

Fig. 4: Results for the baboon input. From top left to bottom right: k=1, k=10, k=20, k=40

Fig. 4 shows the result of the *baboon* input with k={1,10,20,40}. We can get a good idea of the original image with ten elements. With 40 elements the noise considerably decreases.

As the k increases, the amount of details increases as well. For the inputs *baboon*, *peppers* and *monalisa*, k=50 gives a nice result, with a little amount of noise, but for *watch*, this minimum k reaches 200. Fig. 5 shows the results for k=50, including *watch*. Fig. 6 shows the watch with k=50. We can notice the noise in this bigger version. Fig. 7 shows the result for k=200 in the same image as Fig. 6

V. Compression

The algorithm as implemented does not result in a smaller image, nor in a less bit-per-channel image, so, the result images have the same amount of bits per pixel and the length

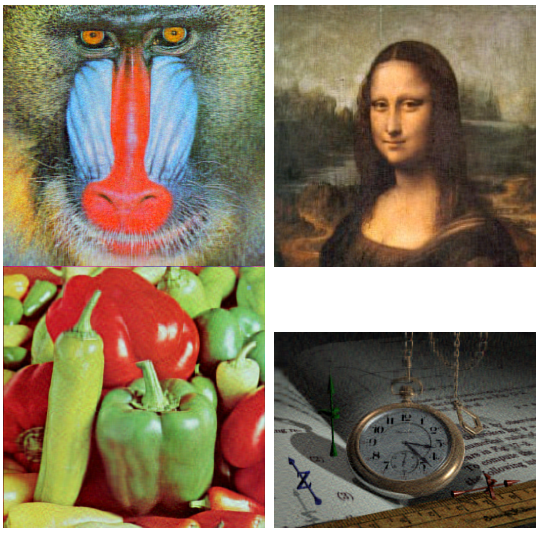

Fig. 5: Results for every input using k = 50.

Fig. 6: Result for *watch* input using the 50 most relevant values.

Fig. 7: Result for *watch* input using the 200 most relevant values.

are equals to the inputs. But the SVD results in an image with less details and more redundant, making it easier to other algorithms to compress. In our case, the images are stored as PNG, and it has a compression without loss. The Table I shows the compression rates for every input, using SVD and k={1,5,10,20,30,40,50}. The TABLE II shows the root mean square errors.

Input	k=1	k=5	k=10	k=20	k=30	k=40	k=50
baboon	0.46	0.61	0.68	0.77	0.82	0.85	0.87
monalisa	0.65	0.85	0.93	1.0	1.04	1.08	1.1
peppers	0.48	0.64	0.71	0.78	0.82	0.88	0.89
watch	0.72	1.01	1.18	1.35	1.47	1.57	1.64

TABLE I: Compression rate per input with k={1,5,10,20,30,40,50}.

Input	k=1	k=5	k=10	k=20	k=30	k=40	k=50
baboon	5.64	4.92	4.72	4.51	4.33	4.21	4.08
monalisa	5.1	3.7	3.19	2.79	2.57	2.41	2.28
peppers	6.04	4.82	4.25	3.64	3.3	3.06	2.88
watch	5.05	4.41	4.14	3.82	3.62	3.46	3.33

TABLE II: RMSE per input with k={1,5,10,20,30,40,50}.

We can notice that as the k gets bigger, the error gets smaller, but the compression rate doesn't always result in a value smaller than one. This is because the result can put too much noise in homogeneous regions that would be more redundant in the original version. The *watch* input results in the worst compression rates, begining with just 28% of gain on k=1 and ending with 164% of the original image's weight for k=50.

VI. Conclusion

Making the principal component analysis can be very useful to reduce the amount of data of a dataset. In Image processing context it can be used to rebuild the image with just the most important information, making a lossy compression. At this project we used SDV to make the image more redundant and making it easier to compress. The SVD calculated the way we did returns the image with the same length e dimensions, resulting in an image with the same amount of bits when represented as an array. But when the image is stored as PNG, the PNG lossless compression reduces the size of images that became more redundant. In the tests we can see that the watch file became heavier after the "compression", because the noise made it less redundant to compress.

But in another contexts, as machine learning, the image could be transformed into an smaller array, containing the most relevant information from the original input and then used to train the system. In [1], an 8x8 image from MNIST was be transformed into a structure with shape 8x2 and yet achieved 0.8479578 of accuracy, compared to 0.864774 using the original image, taking less time and computational power to train the system with 75% less information.

References

[1] Singular Value Decomposition Example In Python, `https://towardsdatascience.com/singular-value-decomposition-example-in-python-dab2507d85a` Cory Maklin, 2019, August

[2] How Are Principal Component Analysis and Singular Value Decomposition Related?, `https://intoli.com/blog/pca-and-svd/`, Andre Perunicic, 2017, August