Image reconstruction SVD vs PCA

Computational linear algebra for large scale problems

02TWYSM

Fatemeh Ahmadvand

s301384

2022-2023

Image reconstruction SVD vs PCA

1 Introduction

This project aims to compare the performance of principal component analysis (PCA) and singular value decomposition (SVD) for the reconstruction of single grayscale images. I tested both methods on three different images with the same dimensions and compared the results considering the accuracy (measured by the *Frobenius norm* of the error) and the execution time.

2 Install and import all the libraries

The only library I need that was not used in the course is the *Pillow* package. It allows me to handle the images better.

In [1]:
```
#pip install pillow
```

In [2]:
```
import numpy as np
import pandas as pd
import PIL
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from PIL import Image
import time
```

3 Data preparation

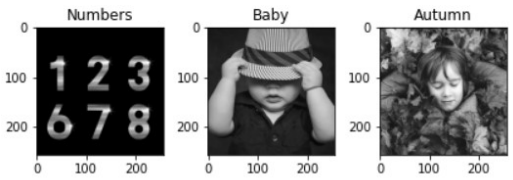
In [3]:
```
imgs = []
imgs.append(Image.open("D:/image/img1.png"))
imgs.append(Image.open("D:/image/img2.png"))
imgs.append(Image.open("D:/image/img3.png"))

n_images = 3
imgs_names = ["Numbers", "Baby", "Autumn"]
fig, axs = plt.subplots(1,n_images)

for i in range(n_images):
    axs[i].imshow(imgs[i])
    axs[i].set_title(imgs_names[i])
    fig.tight_layout()
plt.show()
```

The images are first stored in a square matrix $I \in R^{256\times256}$, where each component $I_{i,j} \in R_3$ is a 3-tuple containing the value for each RGB channel. I then converted each image to a gray scale so that the new matrix is still $I \in R^{256\times256}$, but now each component $I_{i,j}$ is a scalar in the range $[0, 255]$. For simplicity, I then normalized these values to be in the range $[0, 1]$. The smaller the value, the blacker the pixel. The larger the value, the brighter the corresponding pixel will be and the whiter it will be.

In [4]:
```
imggray = []
imgmat = []
fig, axs = plt.subplots(1,n_images)
for i in range (n_images):
    imggray.append(imgs[i].convert('LA'))
    imgmat.append(np.array(list(imggray[i].getdata(band=0)), int))
    imgmat[i].shape = (imggray[i].size[1], imggray[i].size[0])
    imgmat[i] = np.matrix(imgmat[i])
    imgmat[i]=imgmat[i]/255
    axs[i].imshow(imgmat[i], cmap="gray")
    axs[i].set_title(imgs_names[i])
    fig.tight_layout()
plt.show()
```



Each matrix is then stored in a data frame. To illustrate, below is the data frame corresponding to the baby's picture is shown below.

In [5]:
```
imgs_df = []
for i in range(n_images):
    imgs_df.append(pd.DataFrame(imgmat[i]))
imgs_df[1]
```

Out[5]:

	0	1	2	3	4	5	6	7	8	9	...	251	252	253	254	255
0	0.105882	0.109804	0.109804	0.109804	0.113725	0.109804	0.098039	0.098039	0.098039	0.098039	...	0.094118	0.094118	0.090196	0.090196	0.090196
1	0.113725	0.109804	0.109804	0.113725	0.105882	0.105882	0.101961	0.098039	0.098039	0.098039	...	0.094118	0.094118	0.094118	0.094118	0.094118
2	0.105882	0.105882	0.105882	0.105882	0.105882	0.101961	0.098039	0.098039	0.098039	0.101961	...	0.094118	0.094118	0.098039	0.094118	0.094118
3	0.098039	0.105882	0.105882	0.105882	0.101961	0.101961	0.098039	0.098039	0.101961	0.101961	...	0.090196	0.090196	0.094118	0.094118	0.094118
4	0.098039	0.098039	0.098039	0.101961	0.098039	0.101961	0.101961	0.094118	0.094118	0.098039	...	0.090196	0.090196	0.094118	0.094118	0.094118
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
251	0.133333	0.129412	0.133333	0.133333	0.133333	0.129412	0.125490	0.125490	0.125490	0.125490	...	0.098039	0.098039	0.098039	0.094118	0.094118
252	0.125490	0.133333	0.137255	0.137255	0.137255	0.133333	0.133333	0.129412	0.129412	0.125490	...	0.094118	0.094118	0.098039	0.090196	0.090196
253	0.121569	0.129412	0.129412	0.133333	0.137255	0.137255	0.129412	0.129412	0.129412	0.129412	...	0.098039	0.098039	0.098039	0.090196	0.094118
254	0.117647	0.125490	0.125490	0.129412	0.133333	0.133333	0.133333	0.129412	0.129412	0.125490	...	0.094118	0.094118	0.094118	0.090196	0.094118
255	0.121569	0.121569	0.125490	0.125490	0.129412	0.129412	0.129412	0.129412	0.133333	0.125490	...	0.090196	0.094118	0.090196	0.094118	0.094118

256 rows × 256 columns

4 Principal Components Analysis

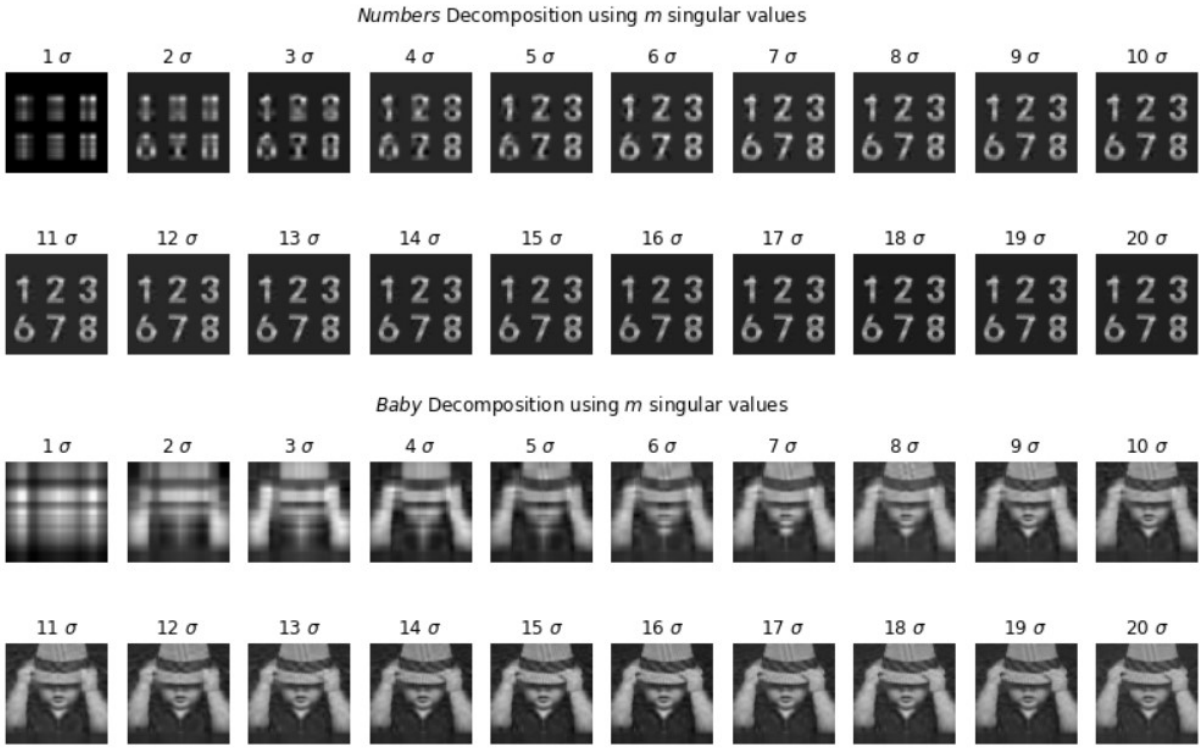
In this section, *sklearn.decomposition.PCA* to be used. For $n \in \{1, 2, ..., 20\}$, PCA with a number of n components is applied to each image. Based on the resulting principal components, each image is reconstructed and the discrepancy with the original image is calculated as the *Frobenius norm* of the difference between the original dataframe and the reconstructed image. Also, for each PCA run, I keep track of the time elapsed for the entire procedure. In general, I expect the image to become more and more similar to the original, even as the elapsed time becomes longer and longer.

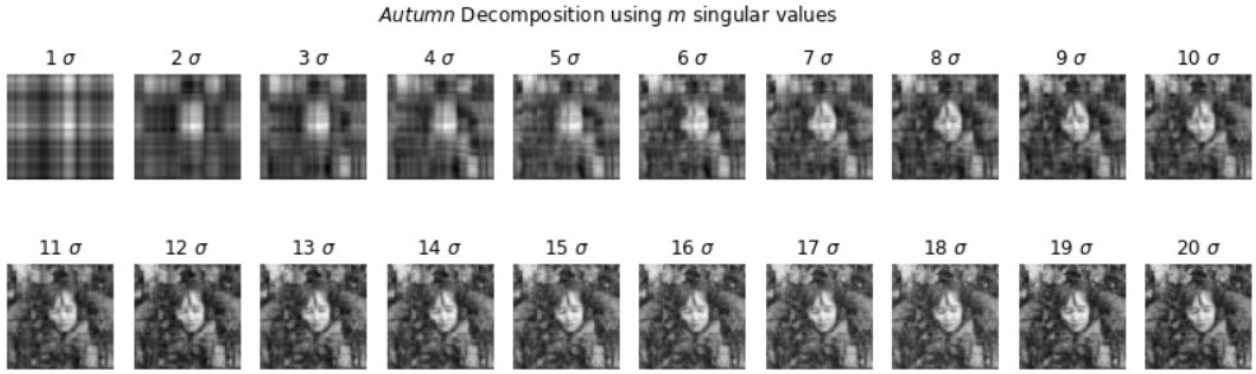
In [6]:
```
pca_imgs = []
pca_error = []
pca_times = []
for i in range(n_images):
    err_PCA=[]
    times_PCA=[]
    pca_img_array = []
    fig, axs = plt.subplots(2, 10)
    fig.set_size_inches(14,4)
    fig.suptitle(f"${imgs_names[i]}$ Recomposition Using $m$ PCs")
    for j in range(1,21):
        ir = (j-1) // 10
        ic = (j-1) % 10
        start=time.time()
        pca_cat = PCA(n_components=j)
        trans_pca_cat = pca_cat.fit_transform(imgs_df[i])
        cat_arr = pca_cat.inverse_transform(trans_pca_cat)
        end=time.time()
        time_elapsed=end-start
        axs[ir][ic].set_title(f"{j} PCs")
        pca_img_array.append(cat_arr)
        axs[ir][ic].imshow(cat_arr,cmap="gray")
        axs[ir][ic].axis('off')
        err_PCA.append(np.linalg.norm(imgmat[i]-cat_arr,'fro'))
        times_PCA.append(time_elapsed)
    #err.append(np.linalg.norm(imgmat/255-cat_arr,'fro'))
    pca_imgs.append(pca_img_array)
    pca_error.append(err_PCA)
    pca_times.append(times_PCA)
    plt.show()
```

5 Singular Value Decomposition

In this section, *np.linalg.svd* is used and applied to each matrix $\tilde{M} = U \sum V$. The time required for the decomposition is also recorded. For each $n \in \{1, 2, ..., 20\}$, each image is reconstructed using n singular values: $reconstimg = U_{:n} \sum_n V_{n:}$ where the double index indicates the number of rows and the number of columns considered (in the case of a":"each row/column is considered). Again, the discrepancy with the original image is calculated as the *Frobenius norm* of the difference between the original matrix and the reconstructed matrix, and the time needed for the whole process is recorded

In [7]:
```
svd_imgs = []
svd_error = []
svd_times = []
for i in range(n_images):
    U, sigma, V = np.linalg.svd(imgmat[i], full_matrices=False)
    err_svd=[]
    times_svd=[]
    svd_img_array = []
    fig, axs = plt.subplots(2, 10)
    fig.set_size_inches(14,4)
    fig.suptitle(f"${imgs_names[i]}$ Decomposition using $m$ singular values")
    for j in range(1,21):
        ir = (j-1) // 10
        ic = (j-1) % 10
        start=time.time()
        reconstimg = np.matrix(U[:, :j]) * np.diag(sigma[:j]) * np.matrix(V[:j, :])
        end=time.time()
        time_elapsed=end-start
        svd_img_array.append(reconstimg)
        err_svd.append(np.linalg.norm(imgmat[i]-reconstimg,'fro'))
        times_svd.append(time_elapsed)
        axs[ir][ic].imshow(reconstimg, cmap='gray')
        axs[ir][ic].set_title(f"{j} $\sigma$")
        axs[ir][ic].axis('off')
    svd_imgs.append(svd_img_array)
    svd_error.append(err_svd)
    svd_times.append(times_svd)
    plt.show()
```





6 Compare the results

As mentioned above, the comparison considers:

1. Accuracy, i.e., the *Frobenius norm* of the discrepancy matrix between the original image and the reconstructed image. Considering all principal components and all singular values, it is possible to construct two accuracy curves, one for PCA and one for SVD.

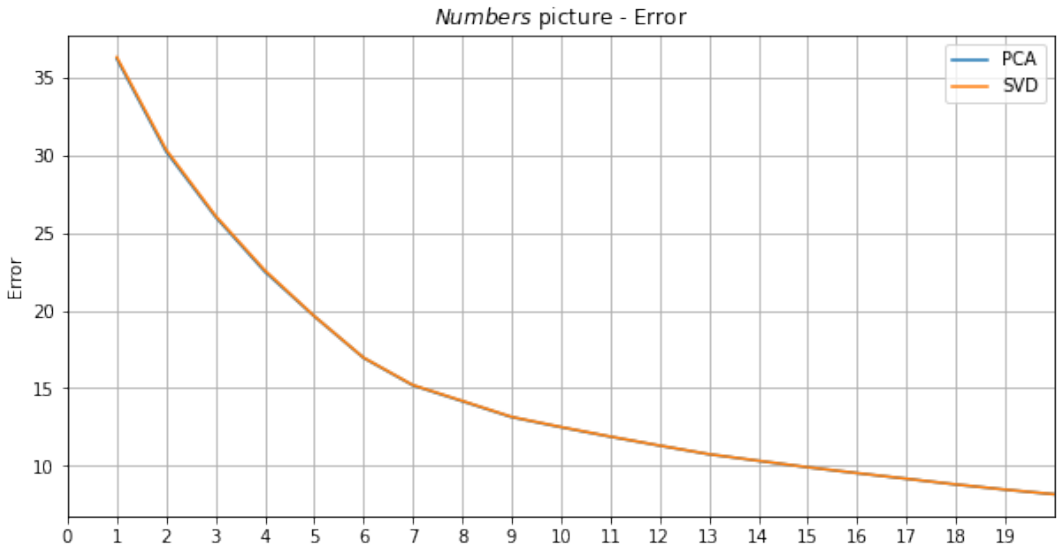
2. Time required for the whole procedure. Again, considering all principal components and singular values, it is possible to create two-time curves, one for PCA and one for SVD.

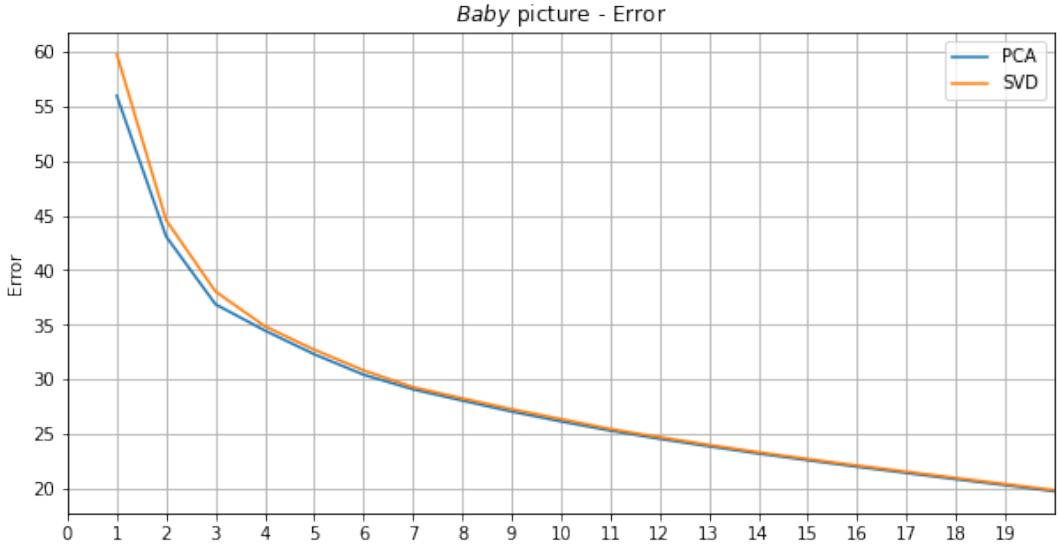
A direct comparison of the two methods is only possible due to the iterative nature of my implementation. This choice was justified because, in principle, a specific comparison between PCA and SVD for a given number of PCs and a given number of singular values are meaningless. In other words, PCA with n principal components is not equivalent to SVD with n singular values, and such a comparison would be meaningless.

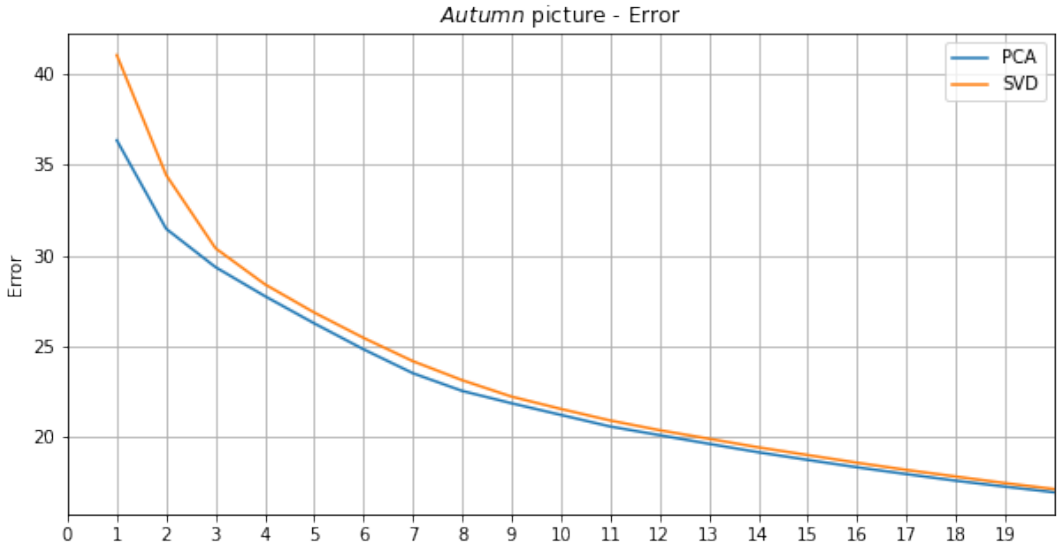
I was not looking for a difference for specific values of n, but for systematic differences estimated by looking at the overall behavior of the two methods across all iterations

0.1 6.1 Accuracy

In [8]:
```
for i in range(n_images):
    plt.figure(figsize=(10,5))
    plt.margins(x=0)
    plt.plot(range(1,21), pca_error[i],label="PCA")
    plt.plot(range(1,21), svd_error[i],label="SVD")
    plt.title(f'${imgs_names[i]}$ picture - Error')
    plt.xticks(ticks=range(20),
               labels=[f'{i}' for i in range(20)])
    #plt.xlabel('Number of principal components')
    plt.legend(loc="upper right")
    plt.ylabel('Error')
    plt.grid()
plt.show()
```

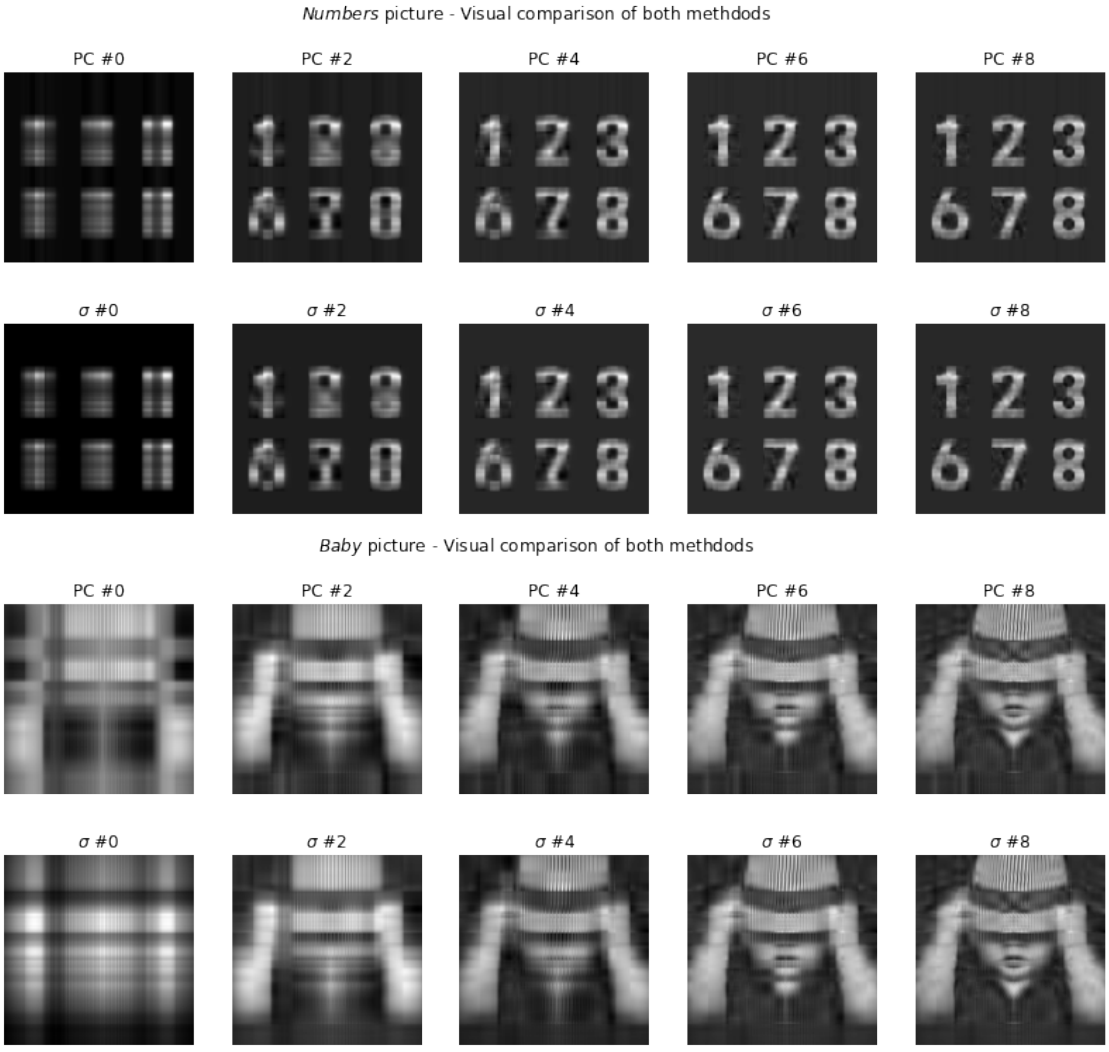

Numbers picture - Error
PCA
SVD
Error
35
30
25
20
15
10
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

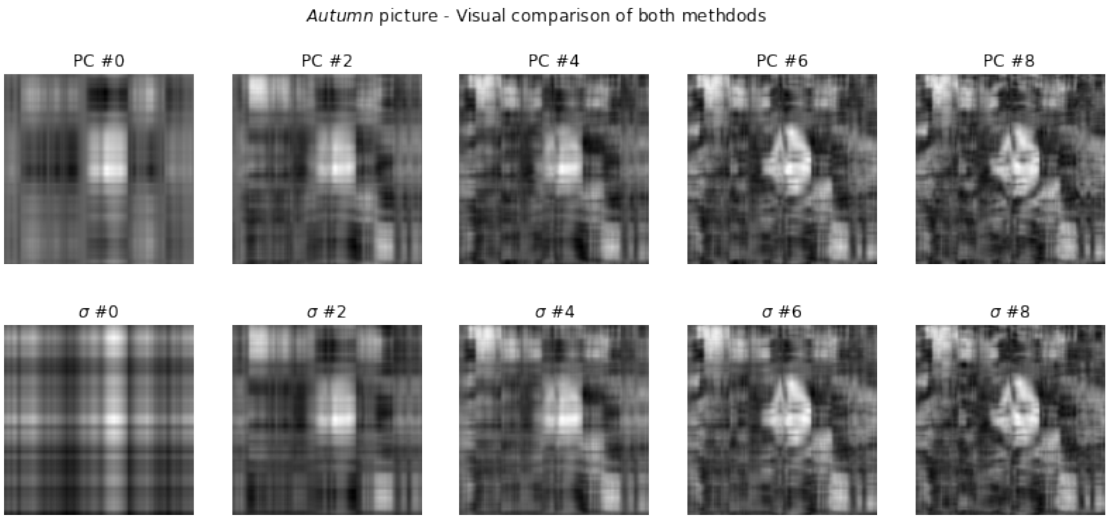

Baby picture - Error
PCA
SVD
Error
60
55
50
45
40
35
30
25
20
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19


Autumn picture - Error
PCA
SVD
Error
40
35
30
25
20
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

0.1.1 6.1.1 Visual comparison

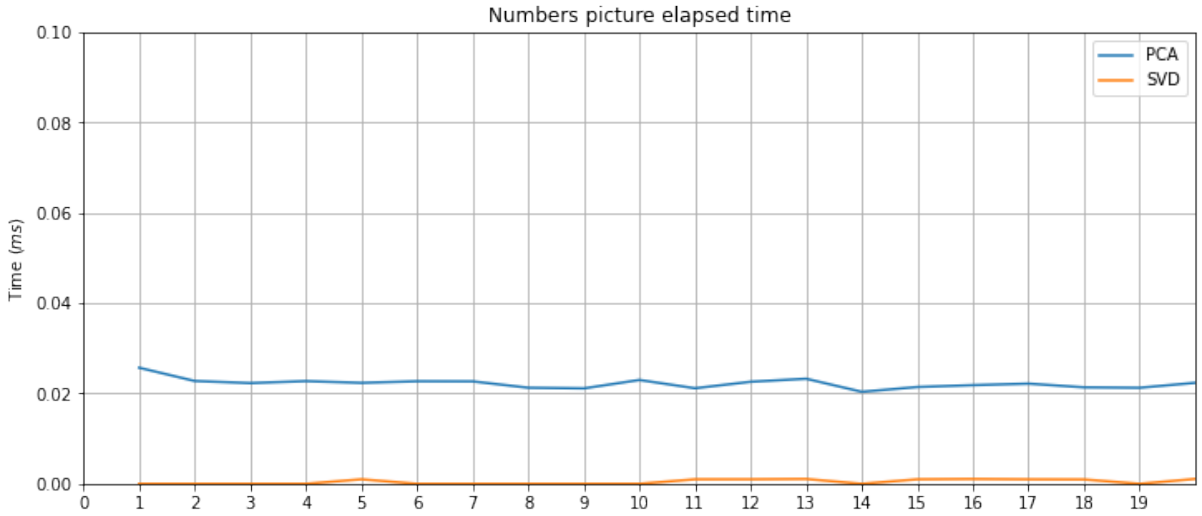
In [9]:
```
for i in range(n_images):
    pca_img_array = pca_imgs[i]
    svd_img_array = svd_imgs[i]
    fig, axs = plt.subplots(2, 5)
    fig.set_size_inches(14,6)
    fig.suptitle(f"${imgs_names[i]}$ picture - Visual comparison of both methdods")
    for j in range(0,5):
        ir = 3*(j // 5)
        ic = j % 5
        axs[ir][ic].imshow(pca_img_array[2*j], cmap='gray')
        axs[ir][ic].set_title(f"PC #{2*j}")
        axs[ir+1][ic].imshow(svd_img_array[2*j], cmap='gray')
        axs[ir+1][ic].set_title(f"$\sigma$ #{2*j}")
        axs[ir][ic].axis('off')
        axs[ir+1][ic].axis('off')
plt.show()
```

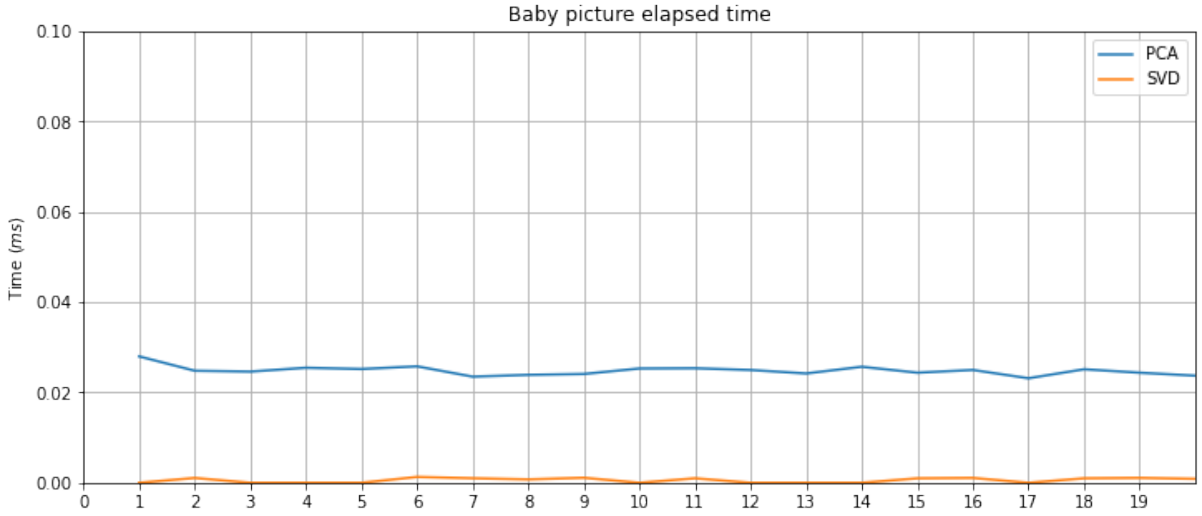


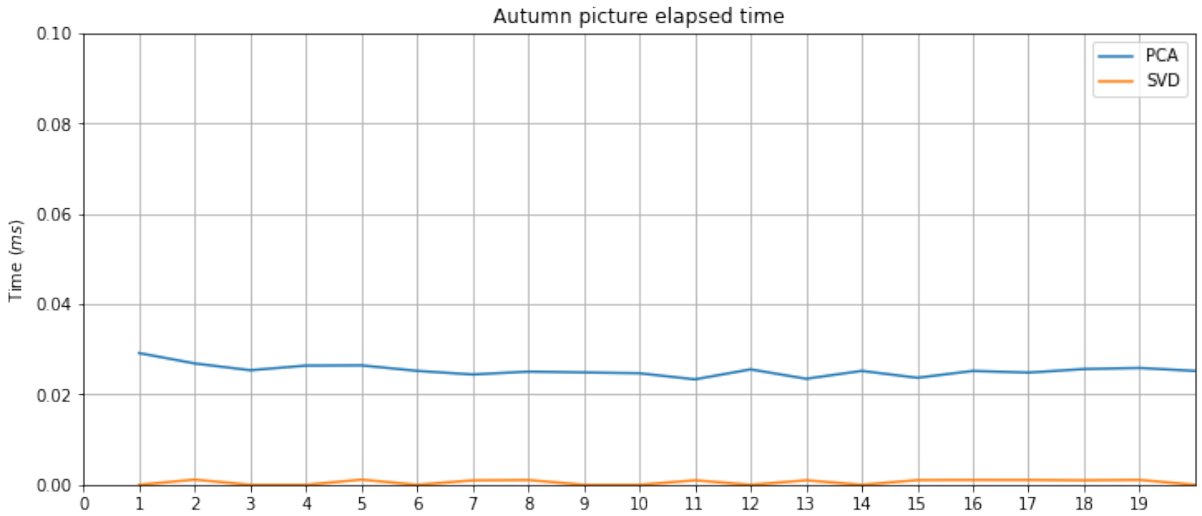


0.2 6.2 Time elapsed

In [10]:
```
for i in range(n_images):
    plt.figure(figsize=(12,5))
    plt.margins(x=0)
    plt.plot(range(1,21), pca_times[i],label="PCA")
    plt.plot(range(1,21), svd_times[i],label="SVD")
    #plt.plot(range(1,51),times_svd_rec,"r--",label="SVD image reconstruction")
    plt.title(f'{imgs_names[i]} picture elapsed time')
    plt.xticks(ticks=range(20),
            labels=[f'{i}' for i in range(20)])
    #plt.xlabel('Number of principal components')
    plt.legend(loc="upper right")
    plt.ylabel('Time ($ms$)')
    plt.ylim(ymin=0, ymax=0.1)
    plt.grid()
    plt.show()
```







7 Result analysis

- Accuracy: I clearly expect PCA to be more accurate than SVD.
  - Numbers's picture: PCA is indeed better, but the difference between both methods is negligible. Not only the *Frobenius norm* of the error is basically the same, but SVD does a better job of reconstructing the background of each image.
  - Baby's picture: here the difference is medium. PCA is numerically more accurate, but the difference is not perceptible to the human eye.
  - Autumn's picture: here the difference between the two methods is big - PCA has higher accuracy, both visually and numerically.
- Total time required: SVD will always be superior in this regard, as it requires fewer operations compared to PCA. Note, however, that I have not considered the time required for the actual factorization. The iterative framework in which I work allows me to perform SVD factorization only once for each image at the beginning of the loop. This justifies why I did not consider

the factorization time when calculating the total time. In my opinion, this is justified as long as an iterative framework is considered.

The image of Numbers is characterized by a substantially sparse matrix ($\sim 60\%$ of the elements are zeros). This makes the calculations easier, and the two methods are completely interchangeable.

The image of the baby is characterized by a denser matrix. The image itself has some drastic color changes, going from light areas (e.g. the baby's head) to darker areas, quite drastically. Therefore, reconstruction of the image with a non-centered technique such as SVD is not particularly satisfactory, and PCA outperforms it.

The image of the fall has a very diffuse pattern where the colors are typically distributed throughout the image, unlike the image of the baby. SVD is characterized by a longer execution time compared to the other methods (though it is still extremely fast), but the lack of defined patterns increases the accuracy, which is almost as good as PCA.

8 Conclusion

Although neither method was pushed to the limit since the images are quite small and in black and white, PCA and SVD are both really solid methods when it comes to the problem of image reconstruction. They took only a few hundredths of a second to produce a more than reliable reconstruction of the images based on a small number of principal components and singular values. However, from the above I can deduce several facts that can help me to find out which algorithm is better than the other:

- SVD generally takes less time, but at the cost of poorer overall accuracy. However, if the input image is particularly dark, the calculations are generally easier. This makes the difference in reconstruction errors between PCA and SVD basically negligible, while the execution time still favors SVD.
- The best overall performance of PCA is achieved when the image is one with few details and a scattered color pattern, like that of the autumn. It still takes longer than SVD, but the difference between the two execution times is the smallest, and the accuracy is still (slightly) better.
- for more complex and structured images (like the image of the baby), PCA may be preferred, as it actually has better "numerical" accuracy than SVD. However, it also depends on the goal of the task. To the human eye, it is quite tedious to actually see a difference between equivalent versions of the two techniques. If the goal is to obtain an image in a short time with no particular detail requirements, then SVD is more than adequate. If, on the other hand, the goal is to get the best possible resolution, then you should go for PCA.

9 Appendix

Hardware: ASUS VivoBook S15 (15-inch, 2021) Processor: 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz 2.80 GHz Memory: 16 GB 2133 MHz LPDDR3 Graphics: NVIDIA SSD: 1 TB