ANALYSES DES DONNÉES

COMPTE RENDU DU PROJET

ENCADRÉ PAR : Mr. CHEFCHAOUNI et Mr. ARRAJI

RÉALISÉ LE : 20 JANVIER 2022

RÉALISÉ PAR :

CHARAF YOUSRI

INTRODUCTION :

Le but de ce mini-projet est de pouvoir appliquer la décomposition des matrices en valeurs singulière et leur application dans le traitement d'image (noire & blanc et RGB) .

Dans la suite, on travaille avec une matrice A (m*n) , tq m>n .

D'après le procédé de décomposition en valeurs singulières, chaque matrice rectangulaire réelle ou complexe (ces coefficients appartient au corps k , tel que K = $\mathbb{R}$ou K = $\mathbb{C}$) peut être décomposée en produit de 3 autre matrices (U , sigma etv) :

La matrice V (n*n) contient un ensemble de vecteurs de base orthonormés de Kn, dits « D'entrée » ;

La matrice U (m*m) contient un ensemble de vecteurs de base orthonormés de Km, dits « De sortie » ;

La matrice sigma (m*n) diagonale et qui contient les valeurs singulière de la matrice transposé(A)*A;

Relation entre traitement d'images et Matrice :

Il faut d'abord savoir que les image peuvent être aussi considérées comme des objets mathématiques , car chaque image (noire et blanc) peut être modélisé par une matrice 2D qui contient des valeurs entre 0 et 255, et chaque nombre est stockés sur 8 bit .

Donc la taille de stockage d'une image est la même de sa matrice

(m*n) , et lorsque m et n prennent des valeurs de plus en plus grandes , le stockage de cette image devient plus difficile (prend beaucoup de mémoire) . Afin de résoudre ce problèmes on utilise la décomposition SVD qui nous permet de réduire la taille de stockage de cette image jusqu'à (k(m+n+1)) , tel que K représente le nombre de valeurs singulière de la matrice transposé(A)*A ;

PARTIE 01 : TRAITEMENT D'IMAGE EN NOIRE ET BLANC

1 / Photo originale de LENA en gris :

Lien pour la télécharger : https://svartzjules.pagesperso-orange.fr/prepa/IPT_spe/lena_gris.png

```
'''    Importation des bibliothéque demandé    '''
from PIL import Image # pour pouvoir utiliser des images ;
import numpy as np  # pour utiliser les arrays prédefinies ;
import numpy.linalg as alg # pour pouvoir utiliser les fonctions de decomposition ;
from matplotlib import pyplot as plt # pour pouvoir afficher les résultats obtenus ;
import matplotlib as mpl
```

```
'''PARTIE 01 : Lire et convertion de l'image , décomposition de la matrice  '''
im=Image.open("lena_gris.png") # pour lire l'image importé ;
T=np.array(im) # pour convertir l'image en matrice ;
m,n=T.shape  # on affect les dimension de la matrice au variable m et n ( m : nombre de lignes , n : nombre de colonnes ) ;
U, S, VT = alg.svd(T,full_matrices=False)  # on decompose la matrice T en fonction de  3 matrices ( U : m*m ; S : m*n ; VT : n*n ) ;
temp = S # on affect la valeur de S a une variable temprelle pour qu'on puisse l'utiliser aprés ;
S = np.diag(S)
```

Explication de l'algorithme de la décomposition en valeurs singulières :

Étape 1 : on cherche le transposé de la matrice (A) , Puis on calcule

Transposé(A)*A ;

Étape 2 : on cherche les valeurs propres puis ces valeurs singulières et on construit la matrice sigma (m*n) en mettant les valeurs singulières (par ordre décroissant) dans la diagonale ;

Étape 3 : on calcule les vecteurs propres, et on construit la matrice

v(n*n) , (Les vecteurs propres représentent les colonnes de V);

Étape 4 : on cherche les vecteurs u (les vecteur colonnes de la matrice V);

Étape 5 : Finalement, on peut écrire la matrice

A = U * sigma * transposé(V) ;

2/ Photo compressée :

```
'''PARTIE 02 : création de la fonction compression '''
def compression(M,k): # la fontion prends comme variable une matrice et le nombre de valeur propres qui doivent rester apés compression ;
    U, S, VT = alg.svd(M,full_matrices=False) # on decompose la matrice en trois matrice ( U , S , VT ) ;
    S = np.diag(S) # la fonction diag retourn une matrice diagonale apartir du vecteur S ;
    M2= U[:,:k] @ S[:k,:k] @ VT[:k,:] # on veut garder juste les k premiéres valeurs singuliére , donc on prend juste les sub_matrixe util
    myimage=Image.fromarray(M2) # on converti la matrice M2 en image ;
    return myimage # on affiche l'image compréssé ;
'''PARTIE 03 : '''
#Plotting the compressed b&w image of Lena
plt.figure()
k=30
fig = plt.figure()
a = fig.add_subplot(1, 1, 1)
imgplot = plt.imshow(compression(T,k))
a.set_title('IMAGE COMPRÉSSÉ DE LEAN EN NOIRE ET BLANC POUR k = {}'.format(k))
plt.show()
```

Exemple d'affichage (pour k = 30):

La dimension de cette image est la même que la dimension de la matrice qui la représente :

m*n = 435 * 395.

Cette image est une version compressée de l'image originale. La matrice qui génère cette image contient que 30 valeurs singulières parmi 395.

Mais on voit que l'image garde encore sa forme avant la compression.

3 / Images de Lena compressée pour des valeurs de K entre (10,…,150) :

```
'''PARTIE 04 : Affichage des images compressés selon les valuer de K '''
j=0
P=[] # on créer une liste vide
for j in range(10,160,10): # on boucle sur les valeurs (10 , 20 ,..., 140 , 150) ;
    P.append(compression(T,j)) # a chaque tour de boucle , on ajoute une image compréssé a notrs liste ;
fig = plt.figure(figsize=(20, 14)) # on cree la figure qui va nois afficher les 15 imges pour des differente valeurs de K ;
columns = 3 # le nombres d'images affichée par lignes ;
rows = 5 # le nombre de ligne d'affichage ;
for i in range(15):
  fig.add_subplot(rows, columns, i+1) # on ajoute un 'sub_plot' a la i_eme position
  plt.imshow(P[i]) #on affiche la i_eme image
  plt.axis('off') #pour ne pas afficher des axes(abs / ord )
  k=(i+1)*10
  compressionRatio = m*n / (k * (1 + m + n)) # la realtion du taux de compression ( uncompressed size / compressed size )
  plt.title("Picture N° {}, Taux de compression {}%".format(i+1, round(compressionRatio, 3 ))) # pour ajouter un titre a chaque image (sub
```

Picture N° 1, Taux de compression 20.677

Picture N° 2, Taux de compression 10.338

Picture N° 3, Taux de compression 6.892

Picture N° 4, Taux de compression 5.169

Picture N° 5, Taux de compression 4.135

Picture N° 6, Taux de compression 3.446

Picture N° 7, Taux de compression 2.954

Picture N° 8, Taux de compression 2.585

Picture N° 9, Taux de compression 2.297

Picture N° 10, Taux de compression 2.068

Picture N° 11, Taux de compression 1.88

Picture N° 12, Taux de compression 1.723

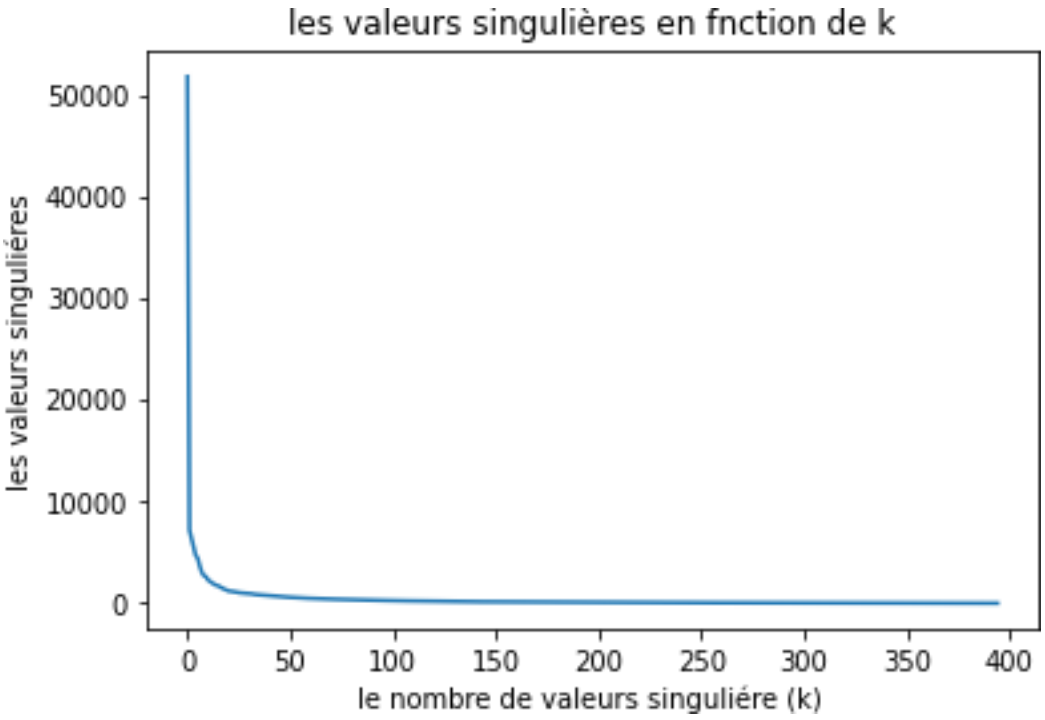
Picture N° 13, Taux de compression 1.591

Picture N° 14, Taux de compression 1.477

Picture N° 15, Taux de compression 1.378

4 / ÉVOLUTION DES VALEURS SINGULÉRES EN FONCTION DE K :

```
'''PARTIE 05 :  afficher le graphe des valeurs singuliéres '''
fig, ax = plt.subplots()  # on veut afficher les valeur singuléres en fonction de k ;
x = [i for i in range(len(temp))] # l'axe des abscisse ;
y = temp # l'axe des ordonnées ;
plt.plot( x , y )
plt.xlabel("le nombre de valeurs singuliére (k) ")
plt.ylabel("les valeurs singuliéres")
plt.title("les valeurs singulières en fnction de k") # pour afficher un titre a l'image
```



D'après le graphe ci-dessous, on remarque que la plus grande valeur singulière est a peut prés 50000, puis la 2eme valeur est proche de 7000, directement après , les valeurs singulières commencent à tendre vers 0 . Ce qui montres clairement que ce qui est le plus important pour une image sont les premières valeurs singulières, et se sont eux qui comporte le plus d'information et de détails.

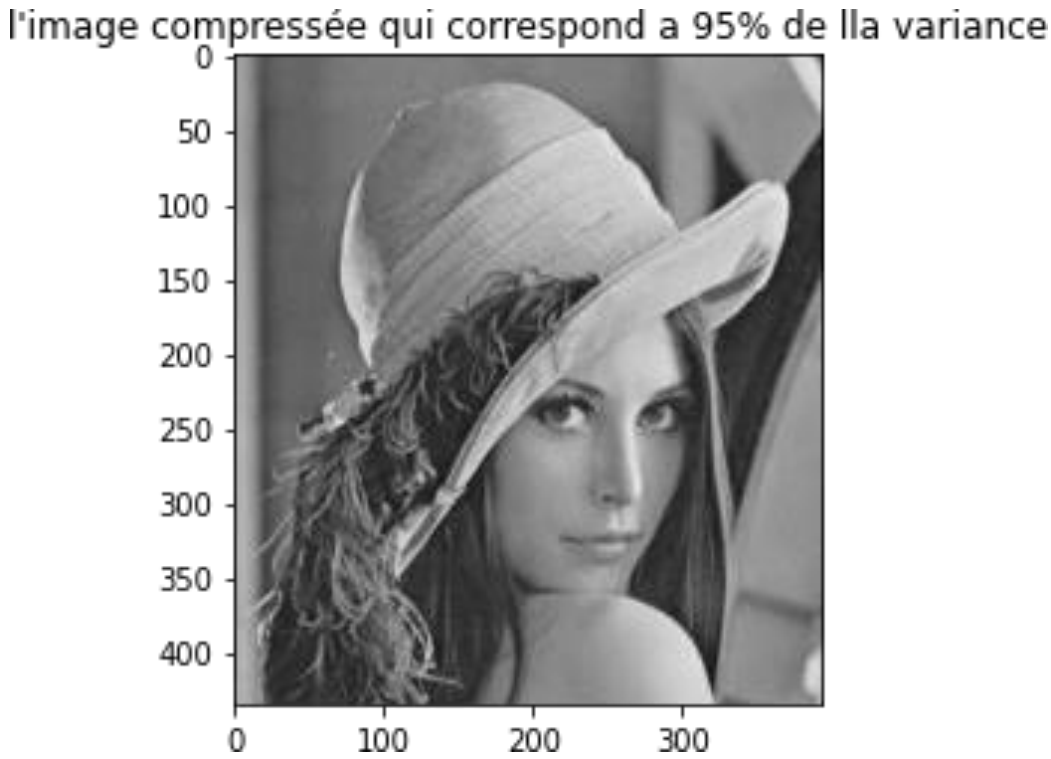
5/ La valeur de cas pour laquelle l'image garde 95% de la variance :

```
'''PARTIE 06 : calcule de k qui donne .95 de la variance '''
for k in range(temp.shape[0]): # on parcour le vecteur qui contient les valeurs singuliéres
  if ( (sum(temp[:k])**2 / sum(temp)**2) >= 0.95): #
    #on cherche le K qui verifie que :
    # [(le carrées de la somme des k 1er valeurs singuliére) / ( le carrées de la sommes de toutes les valeurs singuléres)] > .95
    print("la valeur de k est : " , k) # on affiche le K verifiant la relation ;
    break;
plt.figure()
fig = plt.figure()
a = fig.add_subplot(1,1,1)
imgplot = plt.imshow(Image.fromarray(U[:,:k] @ S[:k,:k] @ VT[:k,:])) # on affiche l'image compréssées qui correspond au 1er k valeurs
a.set_title("l'image compressée qui correspond a 95% de lla variance")
#plt.axis('off')
plt.show()
```

On trouve que K = 241 , puis on affiche l'image compressée qui le correspond

Le taux de compression qui correspond a cette image est de :

Taux de compression = (435 * 395) / 241 * (1 + 395 + 435) = 0.85



Dans cette phase, on cherche à trouver le nombre des valeurs singulières nécessaire pour garder 0.95 de la variance de l'image initiale après compression.

Pour cela, on fait une boucle pour comparer le rapport de la somme des k premières valeurs propres élevées au carrée et la somme de toutes les valeurs singulières élevée au carrées avec 0.95

On trouve qu'on a besoin des 241 premières valeurs singulières.

Lorsqu'on affiche l'image qui correspond juste aux 241 premières valeurs propres, on remarque que l'image est très identique à l'image originale (même si on élimine les 154 dernières valeurs singulières). Ce qui nous permet de réduire la taille de l'image tout en gardant la qualité de l'image initiale.

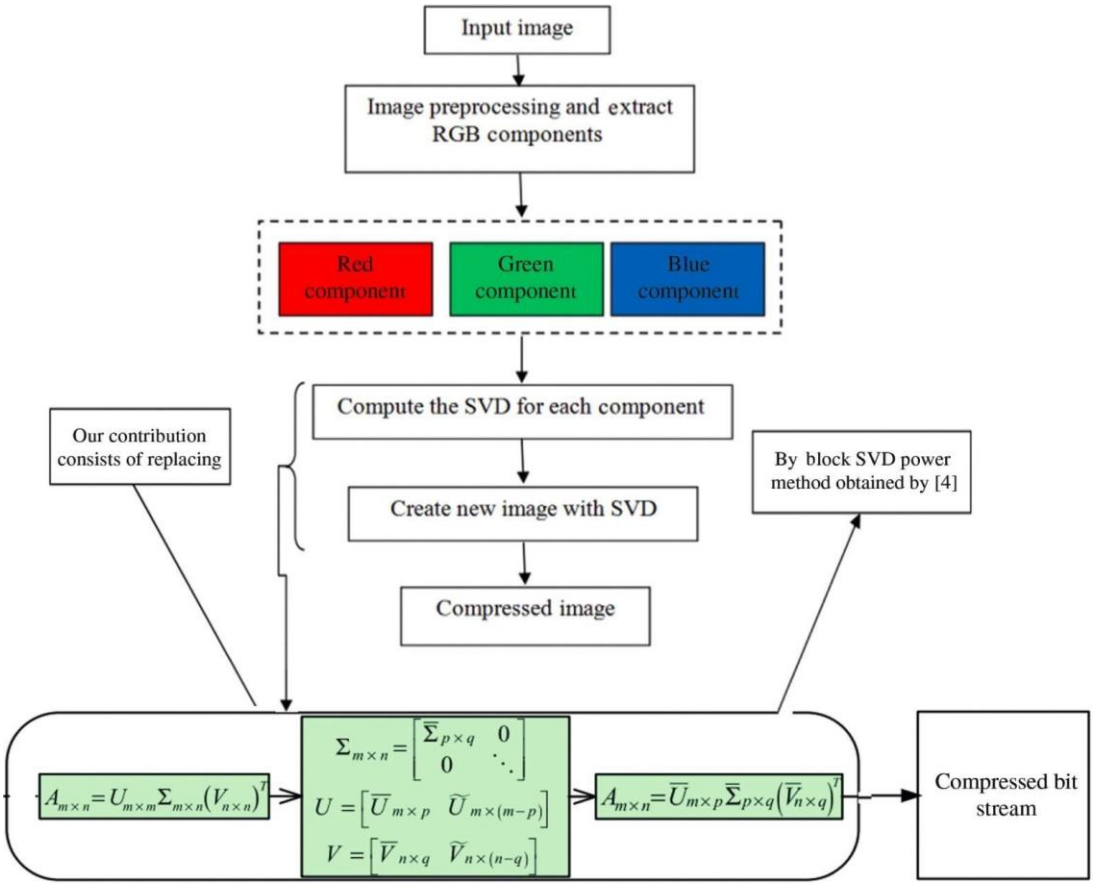
PARTIE 02 : TRAITEMENT D'IMAGE EN RGB

Dans cette partie du projet, on essaie de réaliser un traitement d'image RGB sous format ".png".

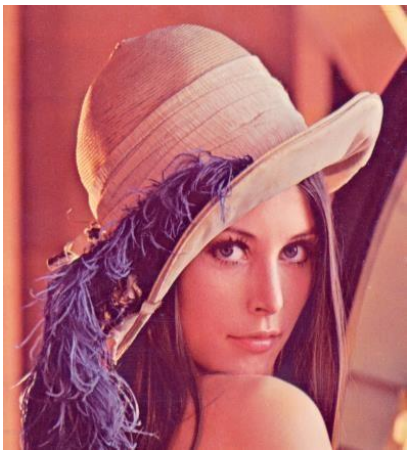
Encore une fois, une image RGB peut être modélisée elle aussi par une matrice, mais la différence que cette nouvelle matrice est 3D (m*n*3) . Donc c'est une superposition de trois matrices, est chaque matrice contienne les informations d'une couche de couleur (rouge, verte et bleu).

En effet, pour pouvoir compresser une image RGB il faut traiter et décomposer chacun des 3 matrices, puis les rassembles toutes les trois pour afficher une image en couleur et compressée.

Le schéma suivant illustre parfaitement ce qui se passe durant la compression d'image en couleur avec SVD :



1/ Photo originale de Lena en couleur :



Lien pour la télécharger : https://svartzjules.pagesperso-orange.fr/prepa/IPT_spe/lena.png

```
image=Image.open("lena.png") # pour lire l'image importé ;
T_couleur=np.array(image) # pour convertir l'image en matrice ;
L,l,h =T_couleur.shape  # on affect les dimenssions de la matrice au L,l,h ( largeur , longueur , hauteur )
U_couleur , S_couleur , VT_couleur = alg.svd(T_couleur,full_matrices=False) # on decompose la matrice en trois autres matrice ;


'''    ploter l'image original de lena en RGB    '''
plt.figure()
fig = plt.figure()
a = fig.add_subplot(1,1,1)
imgplot = plt.imshow(Image.fromarray(T_couleur))
a.set_title('Lena couleur')
#plt.axis('off')
plt.show()
```

2/ Photo compressée pour k = 30

```
''' fonction de compression '''
def compression_couleur(image,k):
  #la fonction prend comme argument une matrice 3d(qui represente une image couleur) et le nombre de valeurs singuliéres qu'on veut garder
  #la foncion retourne une matrice qui represente l'image ne couleur compréssée
  compressed_image = np.zeros(image.shape) # on creer la matrice du retoure qui a les méme dimenssion
  # dans cette fonction , on fait appel 3 fois de la fonction compression , qu'on a utilisé dans la partie du "noire et blanc"
  compressed_image[:,:,0]=compression(image[:, :, 0], k) # on applique la comprssion sur la couche rouge de l'image
  compressed_image[:,:,1]=compression(image[:, :, 1], k) # on applique la comprssion sur la couche verte de l'image
  compressed_image[:,:,2]=compression(image[:, :, 2], k) # on applique la comprssion sur la couche bleue de l'image
  # la matrice 'compressed_image' contient des valeurs negatives et des valeurs supérieur a 255 ,  ce qui affiche des taches verte
  # sur la photo , il faut les éliminer pour avoire une image plus claire;
  compressed_image[compressed_image > 255] = 255 # on réduit toutes les valeurs qui dépasse 255 a la valeur 255;
  compressed_image[compressed_image < 0] = 0 # on ajuste toutes les valeurs negatives on l'affectent la valeurs 0;
  return compressed_image # on retourne la matrice
'''    ploter l'image compressée de lena en couleur avec la valeur de  k = 30    '''
plt.figure()
k=30
fig = plt.figure()
a = fig.add_subplot(1, 1, 1)
imgplot = plt.imshow(Image.fromarray(compression_couleur(T_couleur,k).astype('uint8')))
a.set_title('Image compréssée de Lena en couleur pour la valuer de  k = {}'.format(k))
#plt.axis('off')
plt.show()
```

Dans cette partie de code, on a vu qu'il est nécessaire d'ajuster quelques valeurs de la matrice ' compressed_image ' (pour les valeurs supérieur a 255 on l'affect la valeur 255, et pour les valeurs négatifs on les affects la valeurs 0) avant de la convertir en photo afin d'éviter l'apparition des petites taches vertes dans l'image compressée .

Ce même travail n'a pas été fait pour l'image en noire et blanc parce que ces valeurs aberrantes n'affectent pas sur l'affichage de l'image compressé en mode noire et blanc.

3 / Images de Lena compressée pour des valeurs de K entre (10, 20,140,150) :

```
'''    affichage des images comprésses en fonction des valeurs de k  ainsi que leur taux de compression  '''
# on utilise le méme algoirthme qu'on a utilisé dans l'image en noire et blanc
j=0
l=[] # on créer une liste vide
for j in range(10,160,10): # on boucle sur les valeurs (10 , 20 ,..., 140 , 150) ;
    l.append(Image.fromarray(compression_couleur(T_couleur,j).astype('uint8'))) # a chaque tour de boucle , on ajoute une image compréssé
fig = plt.figure(figsize=(20, 14)) # on cree la figure qui va nois afficher les 15 imges pour des differente valeurs de K ;
columns = 3 # le nombres d'images affichée par lignes ;
rows = 5 # le nombre de ligne d'affichage ;
for i in range(15):
  fig.add_subplot(rows, columns, i+1) # on ajoute un 'sub_plot' a la i_eme position
  plt.imshow(l[i]) #on affiche la i_eme image
  plt.axis('off')
  k=(i+1)*10
  compressionRatio = m*n / (k * (1 + m + n )) # la realtion du taux de compression ( uncompressed size / compressed size )
  plt.title("Picture N° {}, Taux de compression {}".format(i+1, round(compressionRatio, 3 ))) # pour ajouter un titre a chaque image (sub-
```

Picture N° 1, Taux de compression 20.677

Picture N° 2, Taux de compression 10.338

Picture N° 3, Taux de compression 6.892

Picture N° 4, Taux de compression 5.169

Picture N° 5, Taux de compression 4.135

Picture N° 6, Taux de compression 3.446

Picture N° 7, Taux de compression 2.954

Picture N° 8, Taux de compression 2.585

Picture N° 9, Taux de compression 2.297

Picture N° 10, Taux de compression 2.068

Picture N° 11, Taux de compression 1.88

Picture N° 12, Taux de compression 1.723

Picture N° 13, Taux de compression 1.591

Picture N° 14, Taux de compression 1.477

Picture N° 15, Taux de compression 1.378

Dans cette image, on peut visualiser clairement que la qualité de l'image compressée s'améliore au fur et à mesure avec l'augmentation de nombre de valeurs propres restant.

RÉCAPITULATIF :

Tout ce qui précède nous montres l'utilité de l'application de la décomposition en SVD pour le traitement d'image ainsi que la mise en valeur de la décomposition matricielle. Et voici un simple rectificatif pour les principaux intérêts de cette méthode :

Les valeurs singulières représentent l'énergie de l'image, c'est-à-dire que la SVD range le maximum d'énergie de l'image dans un minimum de valeurs singulières.

Les valeurs singulières d'une image ont une très bonne stabilité, c'est-à-dire que quand une petite perturbation (par exemple une marque) est ajoutée à une image, les valeurs singulières ne changent pas significativement.

En plus, la factorisation en SVD est unique.

LES LIMITES DE LA DÉCOMPOSITION SVD :

Il faut mentionner que tous les algorithmes de compressions ont pour but de réduire la taille de l'image de l'image au minimum sans perdre la qualité de l'image initiale. Donc pour la décomposition SVD soit rentable, il faut que le taux de compression (compression ration) soit strictement supérieur à 1. C'est à dire, l'image compresse doit garder un nombre minimale k de valeurs propres tel que : k > (m*n) /(1+n+m) ;

Mais cette condition n'est pas assez suffisante, car on peut trouver un k qui vérifie la relation, et qui nous économise énormément d'espace, mais ne garde pas la qualité de l'image, et vis-versa. Ce qui rend la compression en utilisant la décomposition SVD n'est pas assez pratique.

Pour cette raison, l'utilisation du SVD est assez limités dans le domaine de compression d'image.

PARTIE SUPPLÉMENTAIRE :

Dans le monde de compression (soit d'image, vidéos ..) il existe deux types de compression , LOSSY COMPRESSION et LOSSLESS compression .

QUELLE EST LA DIFFÉRENCE ENTRE LOSSY ET LOSSLESS COMPRESSION ?

On appelle Lossless compression toute compression qui nous permet de récupérer tout la data initiale (avant compression) lors de la décompression de la data compressée.

Par exemple : png (pour les images), RAR/ZIP (pour les fichiers).

Pour lossy compression, après la décompression du compressed data, on risque de ne pas récupérer une data qui peut bien être différente mais suffisamment proches a la data originale pour être utiles d'une certaine manière.

Par exemple : jpeg / jpg (pour les images), mp3 (pour les audios) …

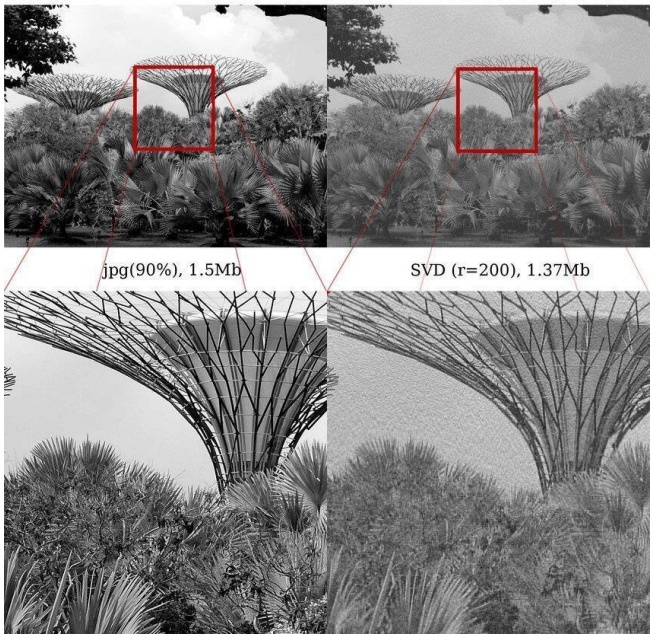
Pour la compression jpeg - une compression LOSSY- , on utilise une autre méthode de compression connue sous le nom de "DCT", cette compression nous permet de réduire beaucoup la taille de l'image en suppriment tous lesdétails non utiles tout en gardant la structure de l'image et sa qualité. Cette compression est beaucoup plus utilisée dans la technologie du web ... Car tout simplement cette compression nous permet d'obtenir une image avec une qualité supérieure avec une très petite taille ce qui rend le
Fonctionnement et l'affichage de site-web beaucoup plus rapide et pratique.

Exemple de compression jpeg :

Donc on remarque clairement qu'on a pu gagner plus de 350 KiB de stockage et en gardant presque la même image.

COMPARAISON ENTRE JPEG ET compression avec SVD :

Un autre exemple pour mieux illustrer la grande différence entre la compression en utilisant la SVD et la compression jpeg :



Ainsi que dans cette photo, on voit vraiment la différence entre l'utilisation des deux méthodes. Même si les deux images on a peu près la même taille (1.4 Mb).