Analyse de Données

Mini Projet SVD avec Numpy

1. Images en noir et blanc et Numpy

Une image grise au format PNG est simplement donnée par la luminosité de ses pixels, qui est un entier entre 0 et 255 (donc encodable sur 8 bits). Pour une image couleur, il y a trois composantes (rouge, verte et bleue) par pixel, ici nous n’aurons qu’une composante car nous traitons d’images en noir et blanc.

Voici comment récupérer les pixels d’une image sous forme d’un tableau Numpy :

from PIL import Image import numpy as np

im=Image.open("lena\_gris.png") #à télécharger sur le site web. Adapter le chemin sous Spyder. T=np.array(im)

h,l=T.shape #hauteur, largeur de l'image

Le résultat est un tableau Numpy, dont les éléments sont de type uint8 (pour *unsigned integer*, entiers non signés, sur 8 bits). T[i][j] donne la valeur du pixel à la *i*-ème ligne, et la *j*-ème colonne, indexées à partir de 0 (le pixel (0*,* 0) est le coin en haut à gauche).

À l’inverse, on peut créer une image à partir d’un tableau Numpy au format uint8 via

Image.fromarray(tableau)

Souvent, le tableau en question est obtenu à partir d’un tableau existant, mais on peut en créer avec np.zeros : avec h et l hauteur et largeur de l’image à créer, on écrirait np.zeros((h,l), dtype="uint8") pour créer un tableau idoine, qu’il suffit de remplir.

Pour afficher une image, il suffit d’utiliser show. Par exemple im.show() via le code situé plus haut vous affiche l’image de Léna.

1. Compression d’une image par utilisation de la décomposition SVD

Une matrice *M* , de taille *n* × *m*, à coefficients dans R admet une décomposition SVD (pour *singular values decomposition*) de la forme *M* = *U* × Σ × *tV* où :

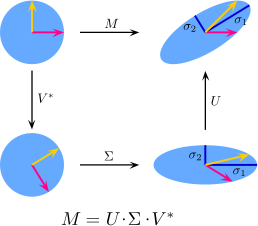
* La matrice orthogonale *V* contient un ensemble de vecteurs de base orthonormés de R*m*, dits « d’entrée » ;
* La matrice orthogonale *U* contient un ensemble de vecteurs de base orthonormés de R*n*, dits « de sortie » ;
* La matrice Σ = (*i,j*)0*≤i≤n−*1*,*0*≤j≤m−*1 est une matrice de taille *n m* « diagonale » (seuls les éléments *i,i* sont non nuls), à coefficients positifs, dont les éléments

×

diagonaux sont les valeurs singulières de la matrice *M* . Les valeurs singulières sont décroissantes : 0*,*0 ≥ 1*,*1 ≥ · · · ≥ *r−*1*,r−*1 ≥ 0 avec *r* = min(*n, m*).

Cette décomposition généralise la diagonalisation des matrices symétriques réelles

positives en base orthonormée, que vous connaissez (dans ce cas, *U* = *V* et les valeurs singulières sont les valeurs propres), d’ailleurs l’existence de la décomposition peut se montrer via le théorème spectral appliqué à *tMM* . L’interprétation géométrique de la décomposition se visualise dans le schéma ci-dessous :



Un moyen simple de compresser une image est d’utiliser la décomposition en valeurs singulières : les grandes valeurs singulières (les premières dans la matrice Σ) sont les plus pertinentes. Si on en garde seulement *k* (virtuellement, on met à zéro les coefficients *sk,k, . . . , sr−*1*,r−*1), on peut se contenter de ne stocker que les *k* premières lignes de *tV* (donc les *k* premières colonnes de *V* ) et les *k* premières colonnes de *U* . À cela on rajoute les *k* valeurs singulières. Ceci fournit donc un moyen de compresser une image.

On peut obtenir une décomposition SVD en Python via la fonction svd du sous-module numpy.linalg, qu’on importera comme suit :

import numpy.linalg as alg

Après avoir lu l’aide de la fonction svd (via help(alg.svd)), écrire une fonction compression(M,k) prenant en entrée une matrice M et un entier *k*, et retournant la matrice *U* Σ*ktV* où Σ*k* est la matrice Σ où les *k,k, k*+1*,k*+1*,* · · · *, r−*1*,r−*1 ont été mis à zéro. Afficher l’image associée pour Lena, pour *k* = 30.

En pratique, on stockerait uniquement les coefficients « utiles » : quel taux de compression obtient-on en fonction de n, m et k ? Application numérique : rajouter l’affichage à l’écran du taux de compression dans la fonction précédente. Faire une boucle pour k ={10, 20, 30, ..., 150}, en affichant l’image obtenue.



Figure 1: Compression via SVD 10, 20, 30, 40, 50, 60}, k = 395 correspond à l’image sans perte, mais déjà avec k = 60 on est très proche de l’image originale.

∈ { }

Tracer le graphe donnant les valeurs singulières en fonction de k. Déterminer la valeur de k permettant de capter 95% de la variance. Afficher alors la compression via SVD pour cette valeur.

On peut appliquer une compression via SVD à une image au format RGB (rouge, vert, bleu). Il suffit d’appliquer l’approche au trois composantes de l’image. Le faire et afficher le résultat obtenu pour différentes valeurs de k.



Figure 2: Compression de Léna avec k = 40.

Lien vers Lena en gris :

[lena\_gris.png (395×435) (pagesperso-orange.fr)](https://svartzjules.pagesperso-orange.fr/prepa/IPT_spe/lena_gris.png)

Lien vers Lena en couleur :

[lena.png (395×435) (pagesperso-orange.fr)](https://svartzjules.pagesperso-orange.fr/prepa/IPT_spe/lena.png)

**Compte-rendu à remettre sous format PDF avec le programme Python joint avant le 33 Janvier 2022 à minuit sur mon mail**

[abdelchafcha@gmail.com](mailto:abdelchafcha@gmail.com)