Université de Bourgogne
Franche-Comté

Projet Algorithmique et complexité

Sujet n°21 : SVD, GSVD, et compression d'images de visages

Auteurs: PINSON Daniel et OGIER Maxime

Année 2022-2023

Contents

1 Introduction

Ce projet consiste à compresser des images à partir d'une technique appelée la Décomposition en Valeur Singulière (ou Singular Value Decomposition en anglais d'où SVD) comme précisé par Hervé Abdi dans [Abdi, 2007]. Le langage imposé est le *Ocaml* et l'utilisation de la librairie *GSL* est autorisée.

Avec des caméras de plus en plus puissantes, viennent des photos qui sont de plus en plus détaillées. Elles contiennent bien plus de pixels, et ont des couleurs bien plus diverses grâce à l'évolution des photo récepteurs. Ainsi, les images prises par ces caméras deviennent de plus en plus lourdes et contiennent de plus en plus d'information, rendant indispensable de les compresser efficacement pour pouvoir les stocker en grande quantité. On propose ici d'étudier une technique de compression qui utilise la SVD.

2 La SVD

La SVD consiste à dire que pour une matrice M dans $\mathbb{K}$ de taille $M \times N$, il existe une factorisation de la forme

$$M = U\Delta V^T \tag{1}$$

avec:

- U de taille $M \times M$, vecteur singulier à gauche pour les valeurs singulières correspondantes.
- Δ de taille $M \times N$ qui est une matrice diagonale dont les coefficients diagonaux sont les valeurs singulières de M, notées δ_k.
- V de taille $N \times N$, vecteur singulier à droite pour les valeurs singulières correspondantes.

Pour effectuer une telle opération, on commence par trouver les valeurs propres de la matrice MM^T, sachant que les valeurs singulières sont les racines carrées de ces valeurs propres. Pour avoir Δ de taille $M \times N$, il faut ajouter des colonnes de 0.
Ensuite on trouve les vecteurs propres de MM^T qui constituent la matrice V, qu'il faut ensuite transposer.
On calcule les colonnes de la matrice U avec:

$$U_k = \frac{1}{\delta_k} M V_k, \quad \{k \in \mathbb{N} \mid k \in [1; n]\} \tag{2}$$

Dans les faits, il n'est pas nécessaire d'effectuer toute ces opérations informatiquement étant donné qu'il existe une fonction dans la librairie *GSL* qui permet d'effectuer une telle factorisation grâce à un simple appel d'une fonction.

3 Compression d'image

La SVD présente des propriétés mathématiques intéressantes que l'on peut exploiter afin de compresser une image qui serait trop lourde. Pour commencer, le rang de la matrice M est facilement identifiable lorsqu'on utilise la SVD. Il s'agit du nombre de valeurs singulières non nulles comme expliqué ici.
Aussi, comme démontré par [Strang, 2003], la SVD donne, pour une matrice rectangulaire M, une approximation optimale qui est de rang inférieur à celui de M. Il s'agit de la meilleur approximation possible par la méthode des moindres carrés.
Ainsi, pour une matrice M de rang r (et qui contient donc r valeurs singulières non nulles), la SVD permet d'approximer celle-ci avec une matrice de rang inférieur K. Pour ce faire, il suffit de garder les K premières colonnes des matrice $U, \Delta \, et \, V$.
On note alors U_K, Δ_K et V_K ces matrices contenant les K premières colonnes. L'approximation de la matrice M vaut alors:

$$M_K = U_K \Delta_K V_K^T \tag{3}$$

La matrice ainsi compressée est dite optimale pour les matrice de rang Kcar elle vérifie l'égalité

$$\|A - A_K\|^2 = trace\{(A - A_K)(A - A_K)^T\} = \min_X \|A - X\|^2$$

pour l'ensemble de matrices X qui sont de rang inférieur ou égal à K, d'après [Abdi, 2007].

La qualité de la reconstruction R_v est donné par le ratio entre la somme des K premières valeurs singulières et la somme totales de toutes les valeurs singulières. On a donc le critère de qualité :

$$R_v = \frac{rang(M_K)}{rang(M)} \tag{4}$$

4 Algorithme et complexité

Notre algorithme est décomposé en plusieurs étapes distinctes avec leur complexité:

1. Ouverture d'une image : $O(MN)$
2. Décomposition en canaux de couleur : $O(MN)$
3. Décomposition SVD : $3O(MN^2)$
4. Padding : $3O(N - M)$
5. Compression des matrices : $3(O(MK) + O(K^2) + O(NK))$
6. Résultat SVD : $3(O(MK^2) + O(MKN))$
7. Suppression Padding : $3O(N)$
8. Recomposition : $O(MN)$

Les librairies camlimages et graphics nous permettent d'ouvrir une image et on estime la complexité de cette étape à un passage entier de cette matrice d'où $O(MN)$, ne serait-ce que pour remplir le tableau. Cette matrice est sous forme de tableau 2D d'entier au format ARGB (0xffffffff).

On décompose ensuite cette matrice en 3 matrices distinctes, Red Green Blue (RGB), qui contiennent chacune la composante en couleur de chaque pixel, cela se fait en un passage de matrice avec la fonction ci-dessous de complexité $O(MN)$.

Listing 1: Fonction de décomposition de couleur

```
let get_colors image =
    let nb_ligne, nb_colonne = (Array.length image), (Array.length image.(0)
        ) in
    let array_image_red = Array.make_matrix nb_ligne nb_colonne 0 in
    let array_image_green = Array.make_matrix nb_ligne nb_colonne 0 in
    let array_image_blue = Array.make_matrix nb_ligne nb_colonne 0 in
    Array.iteri (fun i ligne -> Array.iteri (fun j pixel ->
        array_image_red.(i).(j) <- pixel lsr 16 land 0xff;
        array_image_green.(i).(j) <- pixel lsr 8 land 0xff;
        array_image_blue.(i).(j) <- pixel land 0xff;
    ) ligne) image;
    (array_image_red, array_image_green, array_image_blue);;
```

Les coefficients de 3 devant les prochaines étapes viennent du fait que l'on doit effectuer chacune de ses opérations sur les 3 matrices de couleurs

On utilise la fonction de la librairie *OcamlGSL* qui nous permet de faire la SVD sur une matrice à l'aide de l'algorithme de Golub-Reinsch qui a une complexité de $O(MN^2)$ d'après [Golub and Van Loan, 2013], sur chacune des matrices de couleurs résultante de la fragmentation.
Cependant, la fonction telle qu'implémentée par la librairie GSL n'est applicable que sur les matrices M tel que $m \geq n$. Donc les matrices doivent avoir plus de ligne que de colonnes. Cela pose problème dans notre cas d'application étant donné qu'il nous serais impossible de traiter les photos prises avec une orientation en paysage. On a donc décidé d'effectuer un padding pour ce genre d'image en rajoutant autant de lignes de 0 que nécessaire pour avoir une matrice carrée. On estime le coût de cette opération à $O(N - M)$.

Il faut ensuite appliquer la compression à l'image. Pour cela on défini un taux de compression T_c comme étant le pourcentage de valeurs singulières que l'on souhaite garder. On prend simplement le tableau de la première à la K-ème valeur pour chaque lignes des matrices donnant une complexité de $O(NbLigne \times K)$ pour chaque matrice.

Puis on utilise l'équation 3 pour avoir le résultat de la SVD, ce qui nécessite 2 multiplications de matrices. N'ayant pas trouvé d'informations précises sur la complexité de la fonction Linalg.matmult utilisée dans la documentation de GSL, on estime le coût d'une telle opération à un algorithme naïf où le produit de deux matrices de taille (M, N) et (N, P) est de complexité $O(MNP)$ dans le cas général, car les photos de dimension carrée sont rares.
Ainsi, pour le résultat de la SVD, on effectue 2 multiplications de taille $(M, K)\times(K, K)$ et $(M, K)\times(K, N)$ d'où l'estimation à $O(MK^2)+O(MKN)$.

Dans le cas d'une image en mode portrait, il est nécessaire d'enlever les lignes ajoutées, ce qui se fait en prélevant les N premières lignes, d'où $O(N)$.

Ensuite, on réassemble les 3 matrices RBG pour reconstruire une image compressée, ce qui se fait à l'aide de la fonction ci-dessous, de complexité $O(MN)$.

Listing 2: Fonction de recomposition des couleurs

```
let assign_value r g b =
    let image_compresse = Array.make_matrix (Array.length r) (Array.length r
        .(0)) (Graphics.rgb 0 0 0) in
    let rec assign_aux i j =
        if (i >= (Array.length r)) then
            image_compresse
        else if (j >= (Array.length r.(0))) then
            assign_aux (i+1) 0
        else (
                image_compresse.(i).(j) <- Graphics.rgb r.(i).(j) g.(i).(j) b
                    .(i).(j);
                assign_aux i (j+1)
        ) in assign_aux 0 0 ;;
```

La complexité totale du programme vaut donc

$$\begin{aligned} &3\big[O(MN)+O(MN^2)+O(N-M)+O(MK)+O(K^2)\\ &+O(NK)+O(MK^2)+O(MKN)\big]+3\big[O(N-M)+O(N)\big] \end{aligned} \tag{5}$$

pour une image avec une orientation en portrait, sans quoi il faut enlever le terme $3(O(N-M)+O(N))$.

En simplifiant la complexité, on trouve alors

$$3(O(MN^2)+O(K^2)+O(MK^2)) \tag{6}$$

5 Discussion des résultats

Pour chaque matrice pour laquelle on applique une SVD, il est possible d'avoir la qualité de sa reconstruction. Or, notre programme applique une SVD sur 3 matrices qui ne sont pas l'image originale, mais seulement une partie de celle-ci, donnant ainsi 3 critères de qualités différents. On a donc décidé de donner pour chaque résultat la valeur maximale et minimale de ces trois critères.

La figure 1 est le résultat de notre programme. On observe aucun changement dans l'image à l'œil nu dans la figure 1b. Cela est dû à une grande qualité de reconstruction avec la formule de l'équation 4. Même si on a choisi ici d'enlever la moitié des valeurs singulières de l'image, elles étaient très petites et ne contenaient que très peu d'informations.

Avec T_c plus faible dans la 1c, on obtient une qualité de reconstruction qui est beaucoup plus faible, laissant paraître une forte dégradation de la qualité de l'image.

Aussi, on voit apparaître des artefacts dans l'image. Il sera discuté dans section 6 une piste pour les enlever. On pense que leur apparition est lié à des opérations de conversion de type d'image de ARGB en RGB.

Dans le cas d'une image en mode paysage comme présenté sur la figure 2, on effectue un padding comme précisé dans la section 4. Cependant, il est intéressant de vérifier si celui-ci ne fausse pas nos résultats, en faisant peut-être apparaître les artefacts. La figure 2 permet de réfuter cette hypothèse.

On observe alors qu'il reste les même lignes de 0 qui ont été ajoutée avant l'application de la SVD après celle-ci dans la figure 2b. Il nous suffit donc de prélever les K premières lignes pour pouvoir récupérer l'image compressée.

Il est aussi intéressant d'observer le résultat intermédiaire de la figure 3, obtenu avant d'arriver à ce qui a été obtenu figure 1.

L'algorithme est le même que celui qui a été vu précédemment au détail près que l'on a appliqué la SVD à la matrice originale au format ARGB plutôt que de diviser celle-ci en 3. Ainsi, on constate dans la figure 3b que malgré une grande qualité de reconstruction, l'image est reproduite à la couleur près. La figure 3c montre qu'en gardant le même T_c que lors du résultat final, on a la même qualité de reconstruction, mais les couleurs rendent l'image méconnaissable.

6 Pistes d'amélioration

L'implémentation de la méthode présente certains défauts pour lesquels on propose ici une solution, sans avoir eu le temps de les vérifier.

Calculer la multiplication des 3 matrices de façon asynchrone permettrait d'effectuer ces calculs en parallèles ce qui permettrait d'accélérer les résultats, ce qui est souhaitable étant donné que des images peuvent avoir de très grandes dimensions. Pour donner un ordre de grandeur, la compression de l'image carré vu en figure 1 dure environ 5 s avec sa dimension de 474×474. Alors qu'une image en 612×920 a un temps d'exécution de 25 s. Or, d'après une analyse de marché de ScientiaMobile, les dimensions moyennes des photos prises par les téléphones portables sont de 1080×2400 pixels. Bien que la compression n'est pas une opération effectuée régulièrement faisant qu'un long temps d'exécution est plus acceptable.

L'apparence des artefacts pourrait être expliquée par la suppression du canal Alpha de l'image, qui était au format ARGB lors de l'ouverture, mais qui est convertie en RGB pour les traitements. Ainsi, inclure la composante Alpha de la couleur de l'image permettrait peut-être de remédier aux artefacts.
Une autre piste pourrait être des approximations qui sont faites lors de conversion de flottant en entier. En effet, pour calculer la SVD, il est nécessaire d'avoir des matrices de flottants mais pour afficher une image, il est nécessaire d'avoir une matrice d'entier.
Enfin, une autre explication moins probable serait la précision de l'algorithme de la SVD utilisé. Étant donné que l'on utilise la méthode de *Golub-Reinsch*, or la méthode de *Jacobi* dans la librairie GSL serait plus précise d'après la documentation.

Concernant la complexité du code, celle-ci dépend très largement de 3 termes dans l'équation 5 qui sont issus du calcul de la SVD de la librairie GSL ainsi que le produit matriciel et la compression des matrices.
Concernant le produit matriciel, le postulat est de prendre le pire des cas où l'algorithme n'est pas optimisé, surévaluant l'expression du terme.
Concernant la SVD, il est possible d'apporter une optimisation à l'algorithme de Golub-Reinsch qui a une complexité de $O(N^2 + NM^2)$ dans le cas de matrices avec $M >> N$ d'après [Golub and Reinsch, 1971], ce qui constitue une nette amélioration. Ainsi, dans le cas d'une photo prise en mode portrait, il est possible d'utiliser cette optimisation pour pouvoir accélérer le calcul.

7 Conclusion

Dans ce projet, nous avons donc vu comment se servir de la SVD pour pouvoir compresser une image. La méthode est générique et peut s'appliquer sur tous types d'images, quel que soit leurs dimensions. Seul la nature de l'image est à tenir en compte car une image en RGB et une image en niveaux de gris nécessiteront quelques adaptation pour pouvoir effectuer une compression de la manière la moins coûteuse possible. L'implémentation en couleur étant utilisable pour les images en niveaux de gris.

Concernant le poids des images en Ko, il change peu entre l'image originale et la compressée, sauf pour des images très fortement compressées avec $T_c < 0.1$. Faisant que la compression est efficace seulement si l'on rend l'image presque illisible, rendant cette méthode peu pratiquable telle qu'implémentée.

Aussi, il est intéressant de noter que la compression est irréversible. Donc une image compressée ne peut pas être décompressée, faisant que cette méthode n'est pas utilisable pour le transfert de photos.

References

[Abdi, 2007] Abdi, H. (2007). Singular value decomposition (svd) and generalized singular value decomposition. *Encyclopedia of measurement and statistics*, 907:912.

[Golub and Reinsch, 1971] Golub, G. H. and Reinsch, C. (1971). Singular value decomposition and least squares solutions. *Linear algebra*, 2:134–151.

[Golub and Van Loan, 2013] Golub, G. H. and Van Loan, C. F. (2013). *Matrix computations*. JHU press.

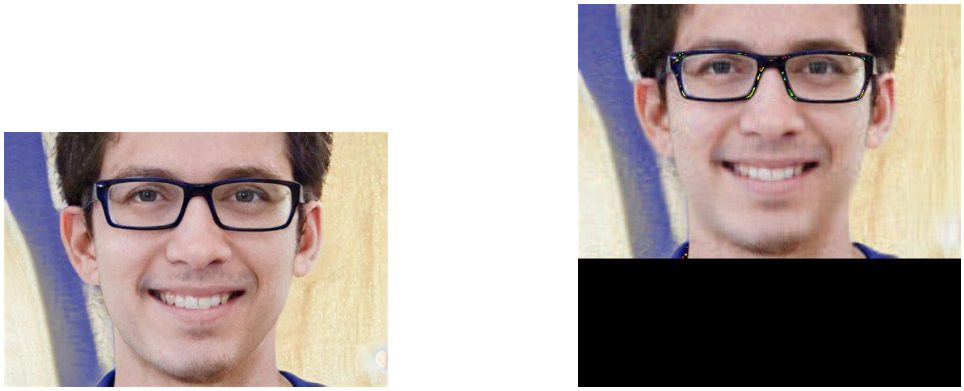
[Strang, 2003] Strang, G. (2003). Introduction to linear algebra, 3rd. *Ed., Wellesley-Cambridge Press, Wellesley, MA*.

(a) Original

(b) $T_c = 50\%$,
$R_v \in [0.9714; 0.9765]$

(c) $T_c = 10\%$,
$R_v \in [0.7170; 0.7617]$

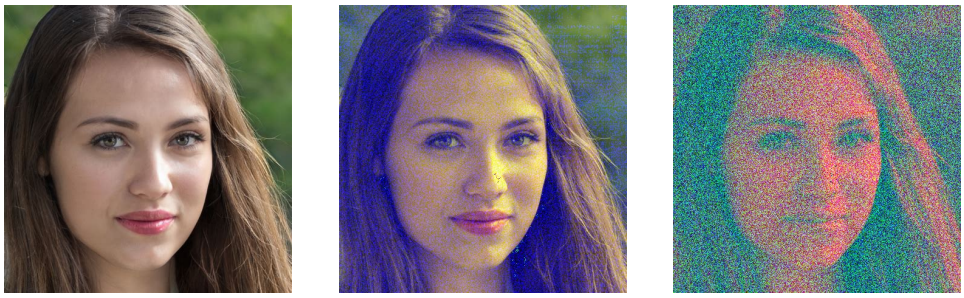
Figure 1: Résultat final sur une image carrée



(a) Original

(b) $T_c = 0.1$,
$R_v \in [0.8174; 0.8458]$

Figure 2: Image en mode paysage sans enlever le padding



(a) Original

(b) $T_c = 0.95$,
$R_v = 0.9999$

(c) $T_c = 0.5$,
$R_v = 0.9765$

Figure 3: Résultat intermédiaire