Sujet : Réduction du poids d'une image

Il est rappelé que le jury attend un exposé de 35 minutes, pédagogique et structuré, fondé sur ce sujet. Ce texte se conclut par une liste de pistes de réflexion pour guider la conception de l'exposé, mais toute initiative personnelle pertinente est appréciée. L'exposé doit contenir une ou plusieurs illustrations informatiques ainsi qu'une discussion autour d'une dimension éthique, sociétale, environnementale, économique ou juridique. Une piste particulière est proposée à cet effet.

Contexte

Aujourd'hui, les envois de photos et plus généralement d'images numériques se font en quantité faramineuse. Tout devient prétexte à l'envoi d'une petite photo sur tel ou tel réseau social. De plus, les téléphones sont de plus en plus performants et permettent à présent de faire des photos d'une grande précision. Tout cela génère donc des flux et du stockage de données considérables.

Un moyen de réduire l'impact de ses échanges d'images est d'utiliser des images avec de moins bonnes résolutions, au risque d'observer un effet de pixellisation. Dans la première partie de ce document, on étudie des moyens de diminuer le nombre de couleurs dans une image et l'effet que cela peut avoir sur le poids de l'image considérée. L'intérêt de cette approche est que l'on garde le même nombre de pixels que dans l'image d'origine tout en permettant de réduire le coût de stockage grâce au format d'image PNG qui sait en tirer parti. La qualité du rendu dépend évidemment du choix des quelques couleurs utilisées.

Lorsqu'une image n'utilise que très peu de couleurs différentes, on peut imaginer qu'elle contienne de grandes plages unies. Dans la seconde partie de ce document, on s'intéresse à la représentation de telles images par des arbres quaternaires qui permettent une représentation concise de ces grandes zones unies.

Au-delà de l'impact sur le poids de l'image, la diminution du nombre de couleurs utilisées dans une image peut être une réponse à des contraintes matérielles. Par exemple, certains systèmes embarqués utilisent des écrans d'affichage en noir et blanc. Dans la dernière partie de ce document, on s'intéresse au cas où on ne dispose que du noir et du blanc. On voit comment reproduire une impression de niveaux de gris en alternant des pixels noirs et blancs et on aborde enfin un traitement d'images plus global permettant de représenter chaque pixel d'une image par un pixel blanc ou noir, mais en propageant les erreurs d'approximation faites au fur et à mesure pour obtenir un meilleur rendu.

Contexte technique

Dans ce texte, on parle d'images en niveaux de gris ou en noir et blanc, mais toutes les approches proposées peuvent être adaptées aux cas des images en couleurs. Les pistes de développement peuvent ainsi être suivies dans le cadre des niveaux de gris, mais il est également possible de tout faire en couleurs. Les images fournies pour les expérimentations sont fournies à la fois en niveaux de gris et en couleurs. Une annexe présente les fonctionnalités Python utiles pour la manipulation de ces images.

On utilise un encodage classique des images. On représente une image par une matrice de pixels dont les dimensions correspondent aux dimensions de l'image. Un pixel est encodé de la manière suivante :

- pour une image en niveaux de gris, un pixel correspond à un entier compris entre 0 et 255, 0 correspondant à un pixel noir, et 255 à un pixel blanc ;
- pour une image en couleurs, un pixel correspond à un tableau de taille 3, contenant trois valeurs `r`, `g` et `b` correspondant aux niveaux de rouge, vert et bleu respectivement, chacune de ces valeurs étant comprise entre 0 (absence de couleur) et 255. Par exemple, le magenta, qui correspond à un mélange de rouge et bleu, sera encodé par $(255, 0, 255)$.

Attention En travaillant avec des entiers non signés sur 8 bits (compris entre 0 et 255), des risques de dépassement d'entiers sont à anticiper lors des opérations arithmétiques. Par exemple, si `n` est un entier non signé sur 8 bits, alors le booléen `n - 128 < 0` ne sera jamais évalué à vrai. Il en est de même pour le test `n + 128 > 255`.

Le format d'images PNG utilise un algorithme de compression pour stocker efficacement les images. L'annexe présente les fonctionnalités permettant d'en-

registrer toutes les matrices obtenues en appliquant les restrictions de couleurs proposées dans ce sujet. Il est alors possible d'observer les effets des différentes approches sur le poids des images.

1 Diminuer le nombre de couleurs

La *quantification* d'une image est le fait de réduire le nombre de couleurs disponibles à une palette donnée. Un tel processus peut être rendu nécessaire par des contraintes techniques, mais il peut également s'avérer très efficace pour réduire le poids de stockage de l'image concernée. En particulier, le format PNG tire pleinement avantage d'un nombre réduit de couleurs dans une image. L'image traitée par quantification garde ainsi le même nombre de pixels, mais son poids diminue drastiquement. Si les couleurs sont bien choisies, on peut ainsi obtenir des images légères en évitant l'effet de pixellisation que l'on observe rapidement en cas de réduction du nombre de pixels.

Une fois la palette de couleurs autorisées fixée, on construit l'image quantifiée en attribuant à chaque pixel, la couleur la plus proche de sa couleur d'origine. Dans le cas d'une image en niveaux de gris, on peut décider de choisir une palette de k niveaux de gris en sélectionnant le noir, le blanc et $k-2$ niveaux de gris répartis à intervalles réguliers entre noir et blanc. Dans le cas d'une image en couleurs, on peut aisément étendre cette approche. Cette solution donne des résultats satisfaisants sur de nombreuses images dès que k est suffisamment grand. En revanche, les résultats peuvent s'avérer décevants si l'image contient de grandes zones dont les nuances de gris sont très proches les unes des autres. Un ciel légèrement nuageux peut ainsi devenir quasiment uni alors que plusieurs des niveaux de gris sélectionnés sont peu ou pas utilisés dans cette image.

Pour pallier ce problème, on peut décider de construire une palette de couleurs autorisées sur mesure pour une image donnée. Pour cela, on peut travailler par itérations successives.

- **Initialisation**

  on fixe une palette (aléatoire, régulièrement répartie ou toute autre palette)

- **Itération**

  - on attribue à chaque pixel la couleur de la palette la plus proche de sa couleur initiale ;

– on modifie chaque couleur de la palette en la remplaçant par la moyenne des couleurs initiales des pixels auxquels on a attribué cette couleur ;

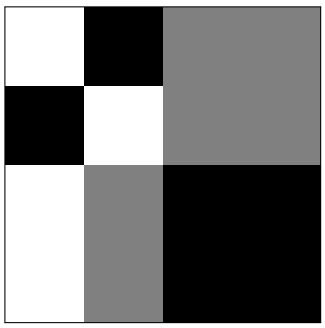
– et on réitère ce processus autant que nécessaire.

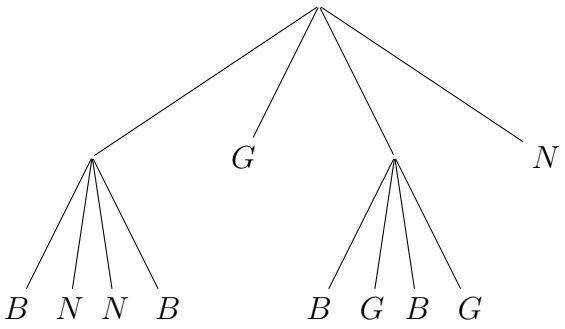
2 Représentation arborescente

Les images obtenues après diminution du nombre de couleurs ont souvent des grandes plages de pixels consécutifs ayant la même couleur. Il peut alors être pertinent d'utiliser des arbres quaternaires pour représenter succinctement ces images.

Un arbre quaternaire est un arbre dont tous les nœuds sont soit des feuilles, soit des nœuds d'arité quatre. Chaque nœud représente une zone carrée de l'image. Les feuilles sont étiquetées par une couleur et représentent une zone carrée unie de cette couleur. Les quatre fils d'un nœud interne sont ordonnés. Le premier fils représente le quart de la zone en haut à gauche, le second représente le quart en haut à droite, le troisième nœud représente le quart en bas à gauche et le dernier fils représente le dernier quart de la zone.

Par exemple, l'image ci-dessous est représentée par l'arbre dessiné à sa droite.





Pour simplifier les expérimentations avec des arbres quaternaires toutes les images fournies sont de dimension 1024×1024.

3 Conversion d'une image en noir et blanc

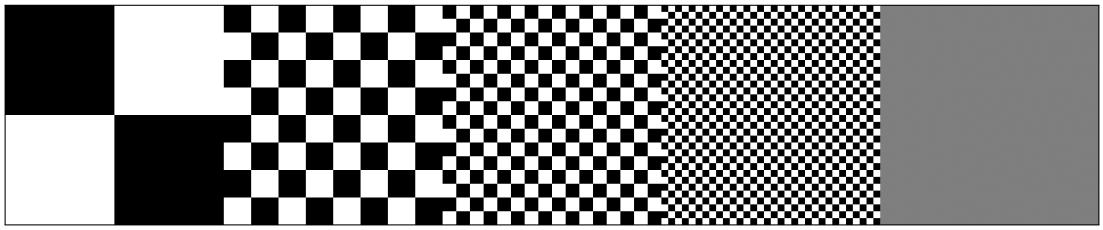
Toujours dans l'optique de diminuer le poids d'une image, ou de répondre à des contraintes techniques, on souhaite développer des approches permettant de représenter des images avec uniquement des pixels noirs et blancs. Une première approche consiste à définir un seuil et à l'appliquer à chaque pixel. Selon si la

couleur initiale du pixel est au-dessous ou au-dessus du seuil, on choisit le noir ou le blanc pour ce pixel. Le résultat peut être un peu grossier, surtout si le choix du seuil n'est pas bien adapté à l'image.

Pour obtenir des résultats plus satisfaisants, on peut essayer d'utiliser des alternances de pixels noirs et blancs pour donner une impression de gris.

3.1 Traitement de zones unies

Si l'on forme un damier en alternant un pixel noir, un pixel blanc, en faisant grandir la taille du damier, on obtient rapidement une impression de gris. De plus, l'impression correspond bien au gris médian. Les images ci-dessous illustrent le phénomène avec des damiers de tailles 2×2, 8×8, 16×16, 32×32 et 512×512.



En utilisant des alternances avec plus ou moins de pixels noirs ou de pixels blancs, on peut obtenir des impressions de plusieurs niveaux de gris possibles.

3.2 Cas général : approche par diffusion d'erreur (*dithering*)

Lorsque l'on applique un seuil, pour un pixel donné, on peut mesurer l'erreur faite dans la représentation de ce pixel (la différence entre la couleur initiale et la nouvelle couleur). L'idée de la diffusion d'erreur est, lors du traitement d'un pixel, de propager un pourcentage de l'erreur faite sur ce pixel sur les pixels voisins qui n'ont pas encore été seuillés. Cette erreur est positive si la nouvelle couleur est noire et négative si la nouvelle couleur est blanche.

Pour appliquer cette diffusion d'erreur, il suffit de parcourir la matrice représentant l'image ligne par ligne et colonne par colonne. Pour la propagation de l'erreur d'un pixel à ses voisins, on peut par exemple utiliser les pourcentages suivants :

- 44% pour le pixel de droite ;
- 19% pour le pixel en diagonale gauche de la ligne du dessous ;

- 31% pour le pixel du dessous ;
- 6% pour le pixel en diagonale droite de la ligne du dessous.

Cette approche correspond à l'algorithme de Floyd-Steinberg.

4 Suggestions pour le développement

On rappelle que bien que le sujet se place dans le cadre des images en niveaux de gris et en noir et blanc, il est possible (mais pas nécessaire) de les adapter aux images en couleurs.

1. *(Section 1)* Expliquer comment étendre la sélection de la palette de niveaux de gris régulièrement répartis au cas des images en couleur.
2. *(Section 1)* La dernière ligne de l'algorithme proposé en Section 1 est imprécise. Discuter le sens de "autant que nécessaire".
3. *(Section 1)* Implémenter la diminution du nombre de couleurs (ou de niveaux de gris) avec une palette régulièrement répartie puis avec la construction d'une palette de couleurs sur mesure pour une image donnée. Appliquer les algorithmes à des images, comparer les résultats obtenus.
4. *(Section 2)* Combien de pixels y-a-t-il au minimum dans une image représentée par un arbre quaternaire de profondeur p ?
5. *(Section 2)* Proposer une implémentation des arbres quaternaires. Implémenter la conversion d'une image en un arbre quaternaire et la conversion réciproque.
6. *(Section 2)* Expliquer comment adapter la représentation d'une image par un arbre quaternaire pour compresser l'image avec perte.
7. *(Section 2)* Expliquer comment stocker de façon synthétique un arbre quaternaire dans un fichier.
8. *(Section 3)* Expliquer comment adapter la représentation du niveau de gris médian avec des pixels noirs et blancs pour représenter d'autres niveaux de gris. Implémenter une ou plusieurs solutions pour évaluer leur efficacité.
9. *(Sections 2 et 3)* Expliquer comment afficher une image donnée sous forme d'un arbre quaternaire de niveaux de gris en utilisant seulement des pixels noirs et blancs.

10. *(Section 3.3)* Implémenter l'algorithme de diffusion d'erreur. Appliquer cet algorithme sur des cas simples pour illustrer le principe.

11. Enregistrer au format PNG les images obtenues avec vos différents algorithmes. Comparer les poids et commenter.

12. Pour compresser une image, on pourrait aussi utiliser des algorithmes de compression génériques tels que Huffman ou LZW. Discuter l'intérêt de l'usage de ces algorithmes dans notre contexte.

13. Historiquement, la communauté informatique a souvent utilisé la même photo pour illustrer les algorithmes de traitement d'images. Cette photo de Lena Forsén, issue du magazine de charmes *Playboy*, représente le visage de la jeune femme et son épaule dénudée. Explicitez les questions que pose l'utilisation récurrente de cette photo dans les travaux et dans certains cours.

Annexe : manipulation d'images en Python

Le fichier `manipulation_images.py` contient trois fonctions dont le prototype et la spécification sont indiqués en commentaire. Elles permettent d'ouvrir, de sauvegarder et d'afficher des images respectivement.

En Python, une image est implémentée par un tableau `numpy`, dont le type est `numpy.ndarray`. Cette structure de données se manipule comme les listes en ce qui concerne l'accès à un élément, la modification d'un élément et l'accès à la taille. Il existe deux différences principales avec les listes :

- tous les éléments d'un tableau `numpy` doivent être du même type ; si on essaie de modifier un élément pour lui donner une valeur d'un type différent, une conversion sera appliquée si possible (ou déclenchera une erreur sinon) ;
- un tableau `numpy` a une taille fixe, définie à la création ; il n'existe donc pas de commande `append` ou `pop` par exemple.

On cite quelques commandes permettant de manipuler les tableaux `numpy`, en ayant importé le module `numpy` par `import numpy as np` :

- `np.array(L)` prend en argument une liste `L` et renvoie un tableau `numpy` correspondant à la conversion de `L` en tableau, et de manière récursive de tous les niveaux de sous-listes ;
- `np.zeros(shape, dtype = type)` prend en argument un uplet `shape` et crée un tableau `numpy` dont les dimensions sont données par l'uplet, contenant uniquement la valeur 0, de type donné par `type`. Par exemple, `np.zeros((1024, 1024), dtype = np.uint8)` permet de créer une matrice correspondant à une image en niveaux de gris dont tous les pixels sont noirs ;
- `np.ones(shape, dtype = type)` comme la commande précédente, mais avec la valeur 1 ;
- si `T` est un tableau, `T + x` renvoie un nouveau tableau correspondant à `T` où on a ajouté `x` à tous les termes ;
- si `T` est un tableau, `T * x` renvoie un nouveau tableau correspondant à `T` où on a multiplié chaque terme par `x`.

Parmi les noms de types pouvant être utilisés par les fonctions `np.zeros` et `np.ones`, on peut citer `int` (entiers), `float` (nombres en virgule flottante) ou `np.uint8` (entiers non signés sur 8 bits, c'est-à-dire compris entre 0 et 255).