Super-Pixels Compte Rendu 1

Souvignet Nathan, Vaillant Hugo

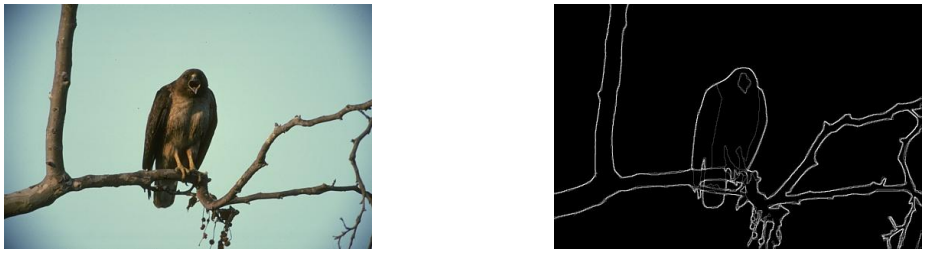
Mars 2025

Contents

1 Nouvelle métrique : Boundary Recall

Nous avons trouvé dans notre article [1] une nouvelle métrique plutôt importante pour les super-pixels. Cette métrique consiste en une vérification des contours de notre image. Pour cela, l'article cite une banque d'images : BSDS500, qui est une banque d'images composée de 500 images naturelles dont les contours ont été tracés par des humains. [3]

1.1 BSDS500



(a) Image réelle (b) Image avec contours dessinés

Figure 1: Image #42049 : Image réelle et celle avec ses contours dessinés par des personnes

Pour obtenir les meilleurs contours possibles, ils ont suivi cette règle : plus des personnes ont associé un même contour, plus ce contour est visible et blanc.

1.2 Contour de nos super-pixels

Une fois que nous avons cette image, nous devons récupérer les contours de notre image super-pixels. Pour cela, nous nous sommes basés sur la norme des gradients dans l'espace RGB.

Pour un pixel de coordonnées (i,j), on calcule la norme de chaque composante (par exemple, uniquement la composante rouge (R)) :

$$R = Pixel[i \times width \times 3 + j]$$

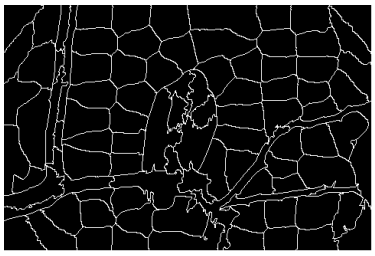
$$xR = Pixel[(i+1) \times width \times 3 + j] - R$$

$$yR = Pixel[i \times width \times 3 + j + 3] - R$$

$$normeR = \sqrt{xR^2 + yR^2}$$

Et si l'une des normes, à savoir normeR, normeG ou normeB, est différente de 0, on la considère comme un contour. Voici les résultats obtenus sur la même image avec différents nombres de super-pixels :

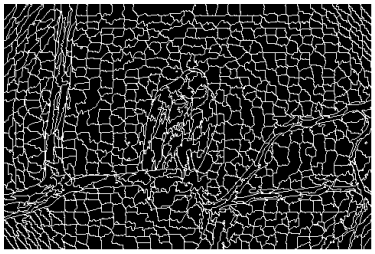
(a) Image super-pixels



(b) Image contours

Figure 2: Image #42049 : avec 100 super-pixels

(a) Image super-pixels



(b) Image contours

Figure 3: Image #42049 : avec 1000 super-pixels

1.3 Calcul du Boundary Recall

Maintenant que nous avons le **ground truth (G)** et **les contours de nos super-pixels (S)**, nous pouvons calculer notre métrique. Pour cela, nous devons compter les vrais positifs (TP) et les faux négatifs (FN) en comparant les deux images.

Pour effectuer cette comparaison de manière efficace, la méthode indique qu'il faut utiliser une approche de voisinage sur nos super-pixels. Cela permet de créer un contour arbitraire autour de chaque super-pixel.

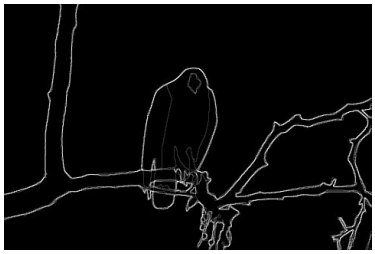
Pour déterminer le nombre de voisins d'un pixel, nous avons utilisé le calcul suivant :

$$(2r + 1) \times (2r + 1)$$

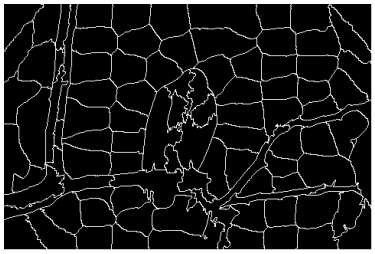
où r est défini comme 0,0025 fois la diagonale de l'image, arrondi à l'entier supérieur.

Cependant, il est précisé que pour le dataset BSDS500, r est égal à 1. Nous avons donc pris en compte les 9 pixels environnants (y compris le pixel central) : $((2 \times 1 + 1) \times (2 \times 1 + 1) = 9)$

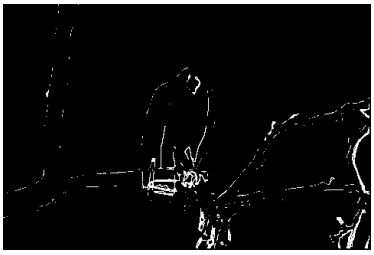
La figure qui suit nous permet d'illustrer le reste des contours dessinés qui n'ont pas été pris en compte lors de notre comparaison.



(a) Image avec contours dessinés



(b) Image contour de super-pixels



(c) Image des faux négatifs

Figure 4: Comparaison de contours

Une fois que nous avons compté les vrais positifs (TP) et les faux négatifs (FN), nous appliquons cette formule :

$$Rec(G,S) = \frac{TP(G,S)}{TP(G,S) + FN(G,S)}$$

Et nous obtenons la valeur de notre métrique. (Pour cette image, mon boundary recall était de 0.693202 avec 100 super-pixels et 0.887283 avec 1000 super-pixels)

2 Compression

2.1 Approche par palette de couleurs

Pour réduire la taille de l'image, nous avons choisi d'utiliser une **palette de couleurs**. Cette approche présente plusieurs avantages :

- Elle permet de réduire la quantité d'informations stockées pour chaque pixel : au lieu de stocker **3 octets** par pixel (R, G, B), nous n'avons besoin que d'un **seul indice** pointant vers une couleur de la palette.
- Le nombre total de couleurs stockées est limité à **k valeurs distinctes** (une couleur par superpixel). Il serait même envisageable de regrouper certaines couleurs proches pour réduire encore la taille, bien que l'impact sur la compression reste à évaluer.

Cependant, un problème se pose : stocker directement un indice par pixel n'est pas optimal. En effet, chaque indice devant être stocké sur **au moins un entier** (`int`, soit 4 octets), cela reviendrait à **augmenter** la taille du fichier au lieu de la réduire (`4o > 3o`).

2.2 Optimisation du stockage des indices

Pour pallier ce problème, nous avons adopté une **codification différentielle des indices** :

- Au lieu de stocker l'indice absolu de chaque pixel, nous enregistrons la **différence** entre l'indice du pixel courant et celui du pixel précédent.
- Cette approche est efficace car, en général, les superpixels voisins ont des indices très proches (la grille initiale étant générée **de gauche à droite, puis de haut en bas**).
- L'unique exception concerne le passage d'une **fin de ligne** au **début de la suivante**, où l'écart peut être important.

Grâce à cette méthode, la plupart des valeurs enregistrées sont **0 ou 1**, avec quelques exceptions où les différences entre deux voisins peut être de 280 pour 5000 régions ou 1580 pour 100 000 régions. Nous utilisons donc des `short int` (2 octets) au lieu d'entiers (`int`, 4 octets). Une amélioration possible serait d'optimiser davantage ces écarts pour permettre l'utilisation de `char` (1 octet). Si les grands sauts d'indices surviennent à des emplacements prévisibles, il devient alors possible d'utiliser plus d'octets pour stocker seulement ces valeurs.

2.3 Compression finale avec Huffman

Une fois les indices différentiels générés, les données SNIC sont compressées avec un **codage de Huffman**. On peut ainsi gagner beaucoup de place si les informations par pixels sont peut variées et se répètent donc fréquemments.

(a) Image de référence

(b) Image compressée

Figure 5: Pour une résolution k = 100 000, avec un PSNR de 30 dB, le taux de compression est de 1,59. L'image compressée occupe alors 496 Kb contre 786,5 Kb pour l'original.

References

[1] David Stutz, Alexander Hermans, Bastian Leibe (2017). *Superpixels: An Evaluation of the State-of-the-Art* https://arxiv.org/pdf/1612.01601

[2] Pablo Arbelaez, Charless Fowlkes et David Martin (2007). *The Berkeley Segmentation Dataset and Benchmark* https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/

[3] Radhakrishna Achanta, Sabine Süsstrunk (2017). *Superpixels and Polygons using Simple Non-Iterative Clustering* https://openaccess.thecvf.com/content_cvpr_2017/papers/Achanta_Superpixels_and_Polygons_CVPR_2017_paper.pdf