# Amélioration de la segmentation d'images d'un dataset

**Numéro du binôme:** 10
**Nom 1:** NAKACHE
**Prénom 1:** Colin
**Email 1:** colin.nakache@u-psud.fr
**Nom 2:** DELPLACE
**Prénom 2:** Alexis
**Email 2:** alexis.delplace@u-psud.fr

**Description du projet et motivation :**

Le but de ce projet est de programmer un algorithme capable d'isoler le sujet d'une image puis de recadrer l'image.

Nous avons choisi ce sujet puisqu'il nous a semblé être le plus intéressant d'un point de vue programmation. De plus, certains autres sujets nous plaisaient moins, notamment par leurs caractères un peu plus mathématique.

La segmentation d'image est importante dans l'analyse des données puisqu'elle permet d'améliorer la qualité des données que l'on donne à l'algorithme, et nous savons que des données de qualitées impliquent de meilleurs résultats [1]. En effet, si la segmentation n'est pas correctement réalisée ou suffisamment performante, elle entraîne une augmentation du taux d'erreur dans l'analyse des données et par conséquent peut fausser celle-ci.

Cela peut donc avoir des conséquences pratiques, comme par exemple la mauvaise détection des piétons par les voitures autonomes, ce qui peut avoir des conséquences dramatiques [2].

Si notre algorithme fonctionne correctement, le taux d'erreur pour un même dataset devrait être plus faible.

**Matériel et méthode :**

La première étape de notre algorithme est d'isoler le sujet de la photo du background, nous avons donc chacun de notre côté codé un programme en python capable de cela.

La première idée d'Alexis fut d'utiliser le filtre de Canny [3] afin de délimiter les contours du sujet, cependant cela n'a pas fonctionné à cause de la complexité du background, en effet les nombreuses feuilles et herbes présentes en arrière plan étaient aussi détectées, ce qui rendait impossible l'exploitation des résultats.

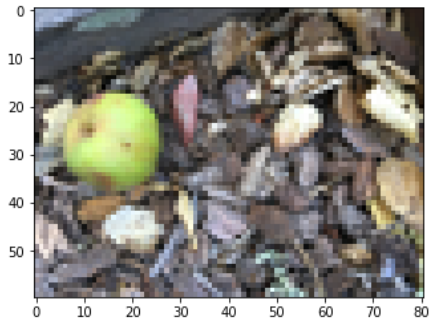


Figure 1 : image “a01.png”

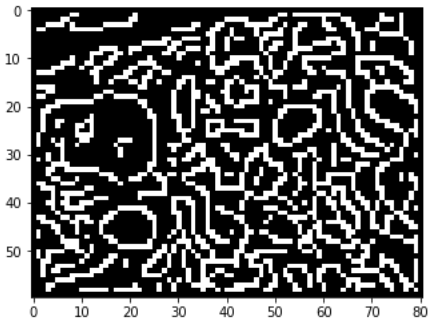


Figure 2 : image “a01.png” après l’application du filtre de Canny

Alexis s’est donc penché sur une autre méthode [4] qui utilise un seuil de niveau de gris (méthode 1), elle consiste à:

1. Flouter une image (Figure 3).
2. Transformer cette image codée en RGB en une image codée en nuances de gris (Figure 4).
3. Fabriquer un filtre qui selon la nuance de gris d’un pixels le transforme en pixel noir ou en pixel blanc (Figure 5).
4. Appliquer ce filtre à l’image originale (Figure 6).
5. Supprimer le bruit par l’application d’un second filtre qui supprime les pixels aberrants (Figure 7).

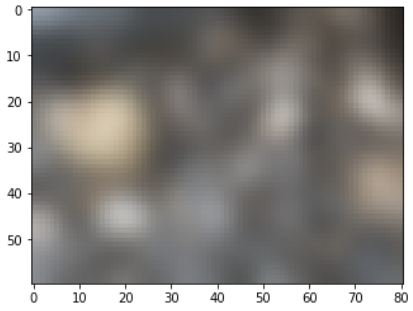


Figure 3

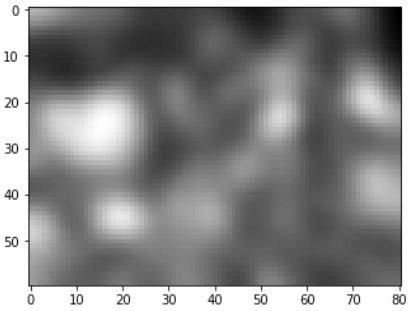


Figure 4

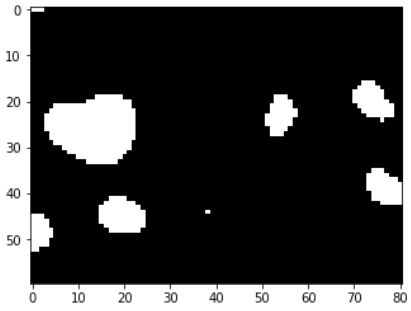


Figure 5

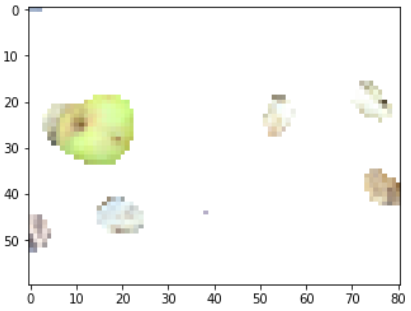


Figure 6

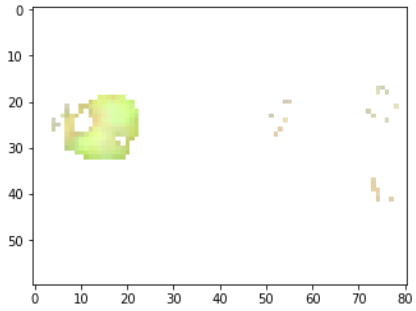


Figure 7

Colin a développé une autre méthode [5] (méthode 2) qui est basé sur un seuil de couleur codé en HSV :

1. Flouter l’image (Figure 10)
2. Changer l’espace de couleurs pour passer en HSV (Figure 11)
3. Fabriquer un filtre qui transforme les pixels lumineux et proches du rouge-jaune-vert en pixels blancs et les autres pixels en noir (Figure 12)
4. Appliquer ce filtre à l’image originale (Figure 13)

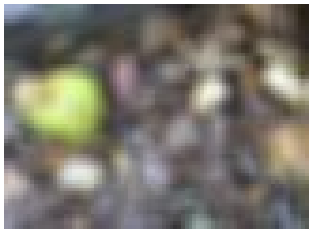


Figure 10

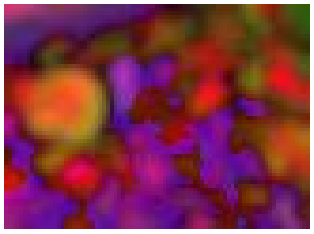


Figure 11

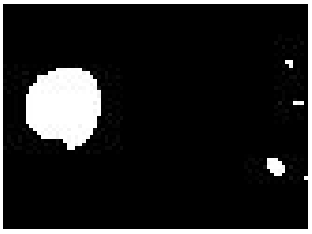


Figure 12

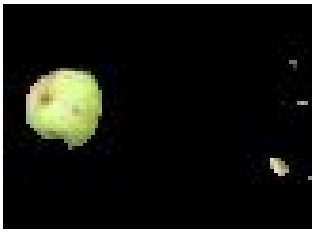


Figure 13

Une fois le sujet isolé du background, nous devons cropper l'image autour de celui-ci, pour cela, nous calculons la position moyenne des pixels restants qui est matérialisée par le point rouge (Figure 8), puis nous croppons l'image autour de ce point (Figure 9).

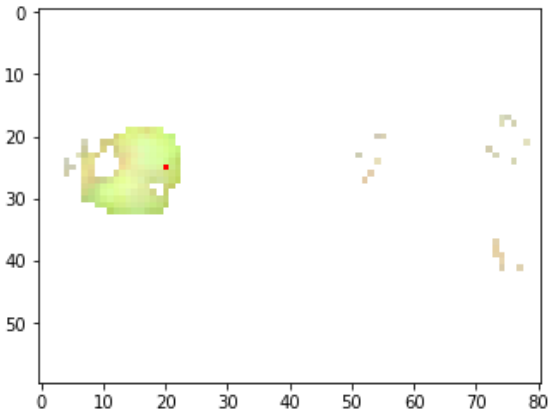



Figure 8

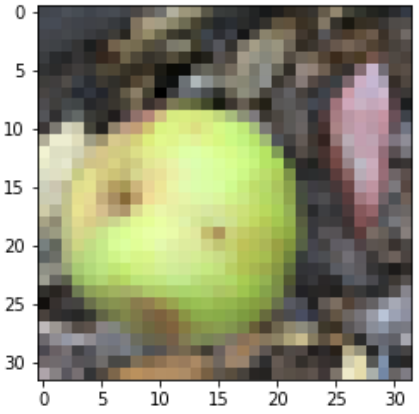



Figure 9

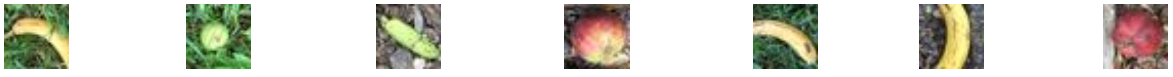
**Résultats :**

Après avoir fait tourner notre algorithme sur le dataset original composé de 333 images de pommes et 158 images de bananes, nous obtenons 3 nouveaux datasets :

Dataset n°1 : "fruit"

Dataset n°2 : "crop_fruit"

Dataset n°3 : "crop"



Après avoir fait tourner l'algorithme des k plus proches voisins (k=3) sur ces 3 datasets, nous obtenons les résultats suivants :

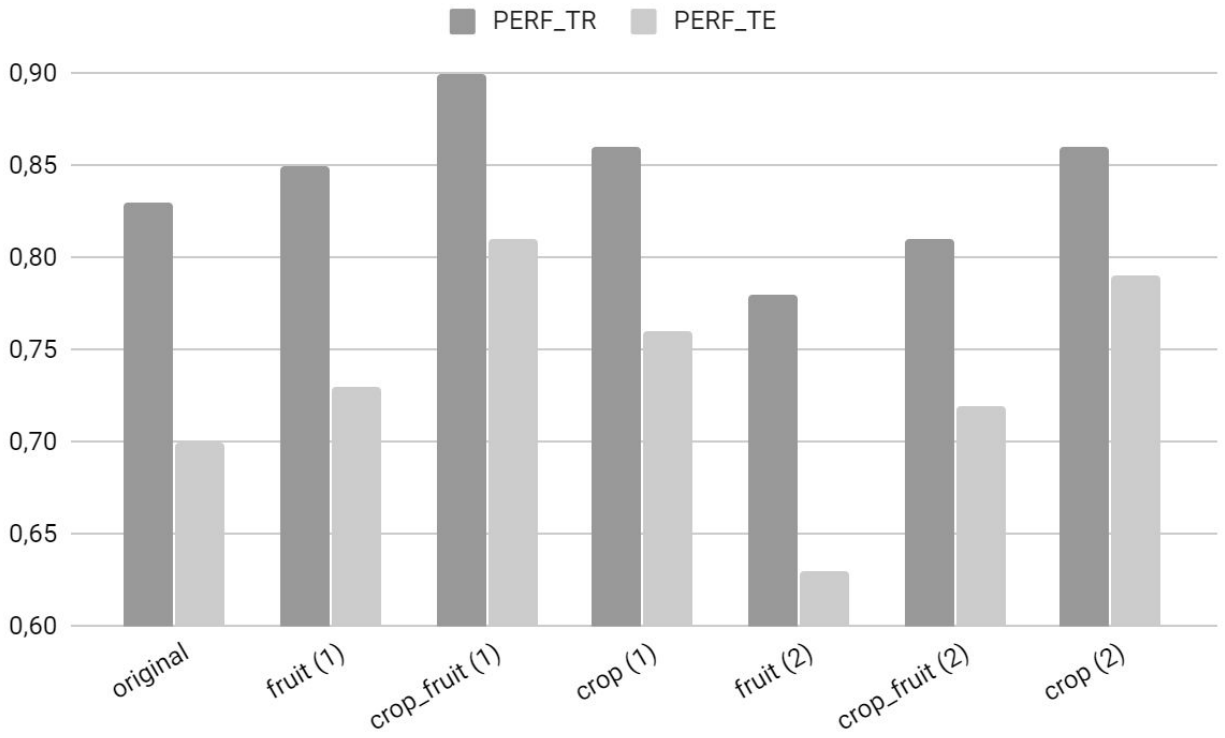



Figure 16 : Graphique des performances des deux méthodes sur les différents datasets (détail en annexe)

On obtient de meilleurs résultats qu'avec le méthode de crop originale pour nos deux méthodes, nous avons préféré garder la deuxième méthode, car elle a de meilleurs résultats que la première. Nous obtenons notre meilleur résultat qui est de 0.90 ± 0.02 sur l'ensemble d'apprentissage et 0.81 ± 0.02 sur l'ensemble de test grâce à la méthode 2 sur le dataset "crop_fruit".

Même si les résultats sont meilleurs, ils ne sont pas parfaits. En effet, sur certaines images un peu complexes, l'algorithme n'arrive pas à déterminer l'emplacement du fruit. C'est le cas par exemple pour l'image "a20".

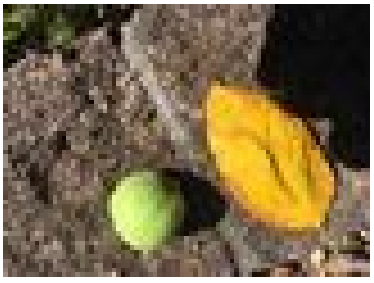


Figure 14

Figure 15

Ce problème est dû au fait que l'on utilise un filtre qui détecte les couleurs vives se trouvant vers le jaune-vert-rouge, tous les éléments proches de ces couleurs sont aussi détectés par le filtre. C'est pourquoi l'algorithme ne fonctionne pas correctement sur les images qui comportent des éléments qui possèdent des couleurs proches de la couleur des pommes et des bananes, ou sur les images où les fruits n'ont pas une couleur assez vive.

**Conclusion :**

Nous avons réussi notre objectif puisqu'en utilisant notre méthode de segmentation pour la classification, nous obtenons de meilleurs résultats sur l'ensemble de test et sur l'ensemble d'apprentissage qu'avec la méthode de crop originale, même si on a vu que certaines images n'étaient pas correctement segmentées.
Cela montre bien que la segmentation est une partie importante dans la classification d'images. Comme nous l'avions prévu, une meilleure segmentation permet d'obtenir de meilleurs résultats.

**Annexes :**

| Méthode n°1 | PERF_TR | STD_TR | PERF_TE | STD_TE |
|---|---|---|---|---|
| **original** | 0.83 | 0.03 | 0.70 | 0.03 |
| **fruit** | 0.78 | 0.02 | 0.63 | 0.02 |
| **crop_fruit** | 0.81 | 0.01 | 0.72 | 0.02 |
| **crop** | 0.86 | 0.02 | 0.79 | 0.02 |

| Méthode n°2 | PERF_TR | STD_TR | PERF_TE | STD_TE |
|---|---|---|---|---|
| **original** | 0.83 | 0.03 | 0.70 | 0.03 |
| **fruit** | 0.85 | 0.03 | 0.73 | 0.03 |
| **crop_fruit** | 0.90 | 0.02 | 0.81 | 0.02 |
| **crop** | 0.86 | 0.03 | 0.76 | 0.02 |

**Références:**


[1] Prince Canuma, “Image Pre-processing”, blog post, October 2018.
https://towardsdatascience.com/image-pre-processing-c1aec0be3edf

[2] Abhinav Sagar, “Mask R-CNN for Self Driving Cars”, blog post, July 2019.
https://towardsdatascience.com/mask-r-cnn-for-self-driving-cars-7d254f3c7b3a

[3] Filtre de Canny, Wikipedia.
https://fr.wikipedia.org/wiki/Filtre_de_Canny

[4] “Image Segmentation”, documentation de Scikit-Image documentation.
https://scikit-image.org/docs/dev/user_guide/tutorial_segmentation.html

[5] Kaustubh Sadekar, “Invisibility Cloak using Color Detection and Segmentation with OpenCV”, Open-CV tutorial, February 2019.
https://www.learnopencv.com/invisibility-cloak-using-color-detection-and-segmentation-with-opencv/