Projet d'IMA201 : Palette automatique de couleur

Manon Heffernan, Cécile Tillerot

20 novembre 2022

Résumé

Dans ce projet, nous avons implémenté la méthode de l'article "Automatic Color Palette" de J. Delon, A. Desolneux, J. L. Lisani and A. B. Petro, permettant de trouver une palette de quelques couleurs à partir d'une image.

1 Quelques méthodes qui sont jugées inefficaces par l'article

Nous avons commencé par tester deux méthodes qui donnent une représentation non-satisfaisante des couleurs dans une image : la méthode de median-cut, et la méthode k-means.

1.1 Median-cut

On raisonne sur chacune des composantes R, G, B. Pour cela, on classe par ordre croissant les listes contenant les composantes R, G, B de tous les pixels de l'image. La méthode de median-cut consiste ensuite à diviser les listes à la médiane, en réitérant ce procédé sur chacun des segments obtenus, jusqu'à obtenir le nombre de segments désirés. On cherche donc une palette de couleurs où chacune représente un groupe de même proportion. Cependant, les petits détails colorés ressortent souvent de la même couleur que le fond, car leur couleur est vite minoritaire devant un fond uniforme. Nous avions fait un test avec l'image *ladybug*, et 64 couleurs ne suffisaient pas pour retrouver le rouge de l'insecte (Figure 1).

Fig.1 : Image originale et image après median-cut ($k = 6$)

1.2 K-means

Nous avons aussi implémenté l'algorithme de k-means, une technique populaire mais qui n'est toujours pas satisfaisante pour notre but. Cette technique de clustering regroupe les points qui sont fortement concentrés autour d'une valeur (cluster). Cependant, elle dépend beaucoup de la position des germes initiaux et donne souvent des couleurs assez fades quand l'arrière-plan est sombre. C'est cette technique que l'on cherche à améliorer en déterminant des germes judicieux.

En Figure 2, on a l'image *bouquet* sur un fond noir, et le résultat de l'algorithme k-means.

FIG.2 : IMAGE ORIGINALE ET IMAGE APRÈS K-MEANS (K = 30)

Ces deux algorithmes sont paramétriques, et ils ne donnent pas des résultats satisfaisants. En effet, il est nécessaire de préciser au préalable le nombre de groupes que l'on cherche, c'est-à-dire le nombre de couleurs. Or, il est difficile d'estimer le nombre de couleurs nécessaire pour représenter une image fidèlement. La méthode implémentée dans l'article que nous avons lu permet de remédier à ces problèmes.

2 Algorithme de l'ACoPa

2.1 L'espace de couleur choisi

L'espace HSI (Hue, Saturation, Intensity) est un espace colorimétrique assez proche dans l'idée d'espaces plus connus comme HSV. Il permet de représenter les couleurs dans un espace plus adapté pour le travail sur des images, car on peut modifier certaines caractéristiques visuelles de l'image (comme le contraste, la luminosité) sans avoir à faire des calculs compliqués sur toutes les composantes RGB de l'image, car ces caractéristiques sont plus directements reliées aux coordonnées dans cet espace colorimétrique.

Nous avons décidé de ne pas utiliser les formules données par les auteurs et autrices dans l'article. En effet, l'inconvénient principal réside dans le fait que H n'est pas calculable lorsque S est trop proche de 0.

$$\begin{aligned} I &= \frac{R+G+B}{3} \\ S &= \sqrt{(R-I)^2+(G-I)^2+(B-I)^2} \\ H &= \arccos\left(\frac{(G-I)-(B-I)}{S\cdot\sqrt{2}}\right). \end{aligned}$$

FIG.3 : FORMULES DE CONVERSION RGB-HSI INDIQUÉES DANS L'ARTICLE

Or, à la suite de recherches sur l'espace HSI, nous n'avons pas trouvé d'exemples utilisant les formules avancées par l'article. Nous avons donc fait le choix d'utiliser les formules les plus souvent utilisées (que l'on trouve par exemple sur Wikipedia), qui offrent notamment l'avantage de ne plus poser de problème sur le calcul de H. En effet, cette dernière utilise arctangente et non plus arcosinus, ce qui permet de ne plus avoir de problème de définition de H, étant donné que arctangente est bornée.

Cela ne nous empêche pas de réimplémenter par la suite le *grey cylinder* en éliminant les valeurs trop faibles dans le calcul de la segmentation de H.

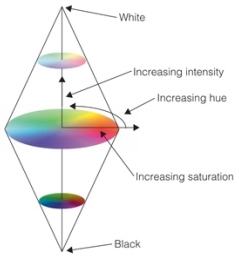

FIG.4 : ESPACE HSI.

Pour notre implémentation, nous sommes obligées de faire une représentation "HSI modifiée", qui comporte les trois composantes H, S et I du pixel, mais aussi les classes dans lesquelles est le pixel pour chaque composante, en fonction des segmentations obtenues. C'est cela qui nous permet de savoir quels pixels considérer pour calculer les segmentations sur la saturation et l'intensité, étant donné que les pixels à considérer pour une coordonnée donnée (S ou I) dépendent des segmentations auxquelles appartiennent les coordonnées précédentes (H pour la segmentation de S, H et S pour la segmentation de I).

2.2 Segmentations et quantifications nécessaires

L'algorithme ACoPa repose sur la bonne segmentation des histogrammes des dimensions H, S et I. Pour cela, nous avons choisi de faire une fonction différente par dimension. On calcule d'abord la segmentation sur H, ce qui nous donne des groupes de pixels avec plus ou moins la même teinte. Puis on fait la segmentation des histogrammes selon S de chaque groupe de pixels. Enfin, sur chaque sous-groupe, on fait une segmentation selon I. C'est ici qu'interviennent les 3 coordonnées supplémentaires ajoutées aux pixels.

On définit pour chaque segmentation une constante Q_hue, Q_sat et Q_int qui donnent la quantification des composantes H, S, I. Comme nous sommes passées dans l'espace HSI avec des formules mathématiques, les grandeurs ne sont pas entières et H, S et I sont respectivement dans [-π, π], [0, 1], [0, 255]. Il faut donc les quantifier pour se ramener à un nombre discret de valeurs pour l'histogramme et diminuer le temps de calcul. On voit par exemple sur la figure 5 pour l'image *teletubbies* comment différentes valeurs de Q_hue, Q_sat et Q_int peuvent influencer les résultats de l'algorithme ACoPa (et avant application de kmeans), surtout si l'on considère la condition du grey cylinder, qui peut exclure complètement certaines teintes de l'image si la saturation est faible et que Q_hue est élevé. Ici, le jaune et le violet sont remplacés par du vert et du rouge, car les pixels de ces teintes n'étaient pas suffisamment saturés.

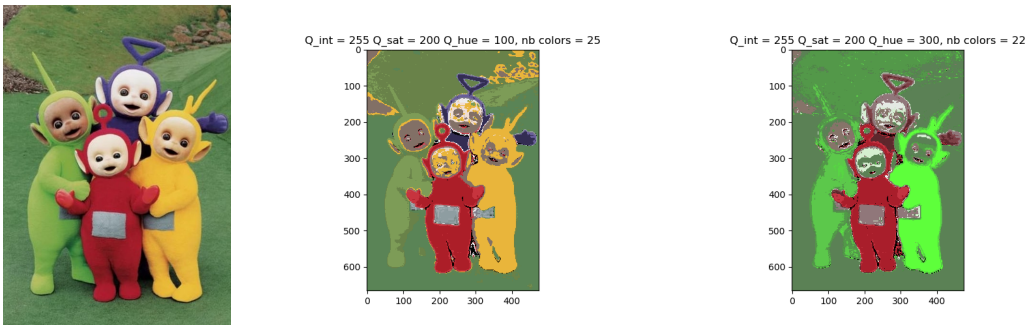

FIG.5 : IMAGES DE TÉLÉTUBBIES : À GAUCHE, ORIGINALE ; AU MILIEU, AVEC Q_HUE = 100 APRÈS FTC ; À DROITE, AVEC Q_HUE = 300 APRÈS ACOPA

Cette segmentation nous permet aussi d'éliminer certaines valeurs de la composante H, définies dans le papier comme un *grey cylinder*. La valeur de saturation pour ces pixels est si faible qu'ils apparaissent tous plus ou moins gris, et le papier recommande de ne pas les prendre en compte dans la segmentation de H. On les rajoute cependant dans le calcul des autres segmentations, en les rattachant au groupe de teintes auquel ils appartiennent, sans les avoir pris en compte pour calculer ces groupes. Voici l'effet de cette limitation sur une image avec des couleurs peu saturées, en figure 5. Appliquer cette condition peut permettre aux couleurs uniformes et peu saturées de l'arrière-plan d'être moins présentes, et de laisser de la place aux couleurs plus vives et saturées du premier plan (la peau de la jeune femme par exemple). La différence est subtile, mais notable

dès l'étape de l'ACoPa.

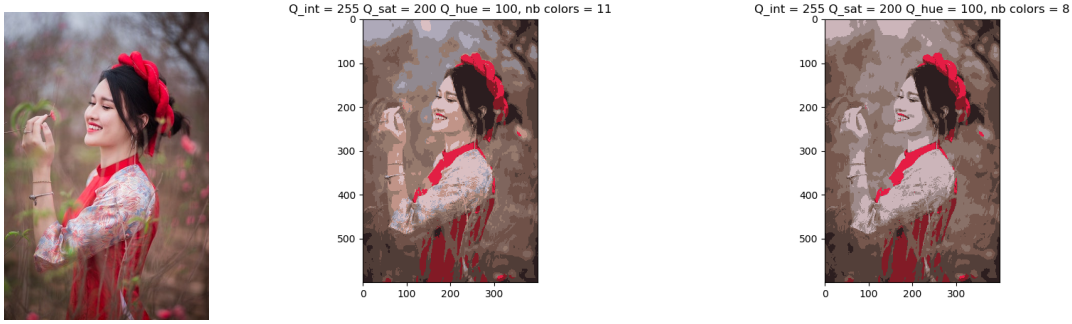

FIG.6 : INFLUENCE DU GREY CYLINDER : À GAUCHE, L'ORIGINAL ; AU MILIEU, AVEC UNE CONDITION QUE S SOIT SUPÉRIEUR À Q_HUE + $20/2\pi$; À DROITE, SANS CONDITION SUR LE GREY CYLINDER

2.3 Le principe de l'algorithme de FTC

Il faut appliquer cet algorithme sur l'histogramme de chaque dimension.

L'algorithme de segmentation "Fine To Coarse" (FTC) permet de diviser un histogramme en plusieurs groupes de pixels tel qu'il est acceptable de dire que chaque groupe suit une loi unimodale. On part d'une segmentation dite "fine" (on segmente au niveau de tous les minima locaux), puis on essaie de regrouper deux segments adjacents en regardant si l'histogramme se rapproche de son estimateur de Grenander (qui lui est unimodal).

Nous devons envisager tous les cas possibles pour regrouper les segments, et calculons donc deux estimateurs : un croissant pour le côté gauche du maximum, et un décroissant pour le côté droit du maximum. L'estimateur de Grenander transforme un histogramme non monotone en histogramme croissant (ou décroissant). Nous avons utilisé l'algorithme des *Pool Adjacents Violators* pour le construire : pour l'estimateur croissant par exemple, on parcourt le segment et si on arrive à un point qui est inférieur au précédent, on le met au même niveau, et de même jusqu'au prochain point qui continue à augmenter. Cet estimateur est notre distribution théorique (croissante ou décroissante), et le test statistique détermine si l'estimateur n'est pas aberrant, c'est-à-dire si le segment de l'histogramme peut être considéré croissant puis décroissant sur une partie, et donc si on peut regrouper les segments et continuer à essayer avec les segments alentours. On s'arrête lorsqu'il n'est plus possible de regrouper des segments en ayant leur union suivant une loi unimodale.

En figure 7, nous avons réalisé quelques tests d'implémentation de l'algorithme des Pool Adjacent Violators sur des histogrammes simples.

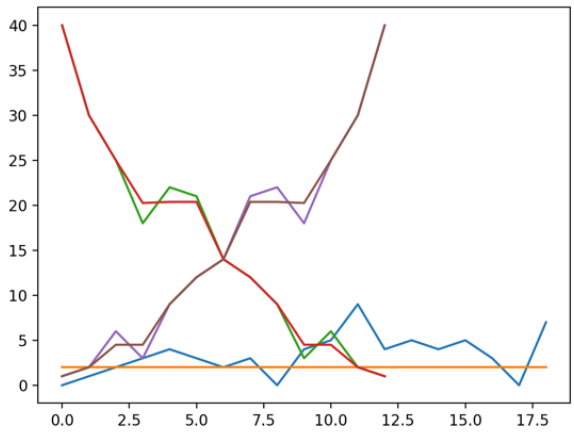

FIG.7 : DES HISTOGRAMMES (VERT, VIOLET, BLEU) ET LEURS ESTIMATEURS DE GRENANDER (ROUGE, MARRON, ORANGE RESPECTIVEMENT)

2.4 Le test d'unimodalité

Selon notre implémentation, on trouve que l'hypothèse d'unimodalité est vraie si le résultat du test statistique rapprochant l'estimateur de Grenander décrit dans la section suivante est 0. Il reste à choisir et implémenter ce test. Dans le papier, il est dit que la solution optimale est un test multiple, qui teste plus d'une hypothèse. Mais un test du khi-deux peut aussi convenir pour déterminer si un histogramme est la réalisation d'une loi.

Les hypothèses sont :

H_0 : L'histogramme peut être vu comme la réalisation de la loi que suit l'estimateur de Grenander.

H_1 : L'histogramme ne peut pas être vu comme la réalisation de la loi que suit l'estimateur de Grenander.

Et le test statistique :

$\delta(T) = 1$ si $T(X) \geq C$ avec $C = q_{\chi^2}(1-\alpha)$ et $\delta(T) = 0$ sinon, où $q_{\chi 2}$ est la fonction quantile de la loi $\chi^2(k)$, α choisi à 0.01, un niveau standard du test du khi-deux.

T est l'estimateur de la loi du khi-deux calculé à partir de l'histogramme et de l'estimateur de Grenander. Sa formule est :

$T = \sum_{j=1}^{len(histo)} \frac{(histo[j]-estimateur[j])^2}{estimateur[j]})$

Avec ce test et les sections précédentes, nous avons tous les élements nécessaires pour implémenter l'algorithme de FTC, et pour déterminer une segmentation selon les principaux modes des histogrammes selon H, S et I.

3 Détermination de la palette de couleurs

3.1 Calcul des couleurs représentatives, et images obtenues après l'algorithme de l'ACoPa (sans k-means)

En ayant segmenté les trois histogrammes selon H, S, et I, les uns dépendemment des autres, on sait classer les pixels dans un groupe de pixels qui est unimodal. Pour afficher une image et voir la palette de couleurs proposée avant k-means, il faut donc déterminer quelle valeur on donne aux pixels d'un même groupe (de H, de S ou de I). C'est le rôle de nos fonctions "compute mode", qui repèrent le maximum des valeurs des pixels du segment et qui choisissent cette valeur pour représenter la classe.

Pour afficher l'image, on a plus qu'à associer à chaque pixel les valeurs sur H, S et I qu'il doit prendre selon sa classification. On obtient alors des images plus satisfaisantes que dans les méthodes précédentes, avec moins de couleurs. Par exemple, pour *ladybug* (voir chapitre 1), le rouge n'est plus perdu, tout en ayant seulement 10 couleurs sur l'image.

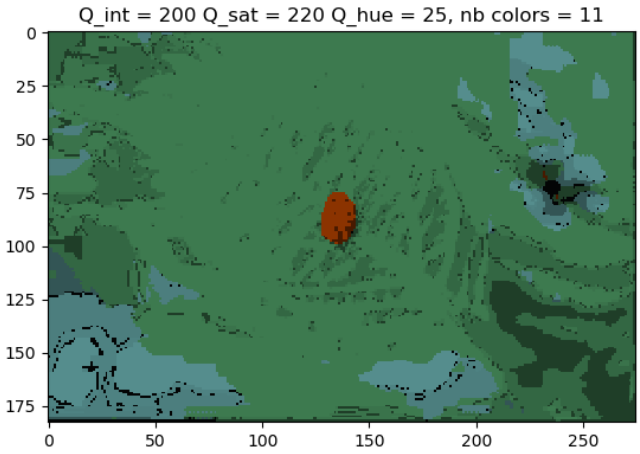

FIG. 8 : RÉSULTAT DE LA FONCTION ACOPA POUR L'IMAGE *ladybug*

Cependant, une image avec des objets de couleurs assez proches peuvent se retrouver perturbées par l'opération : dans l'exemple de la figure 9, on ne distingue plus le plateau en bois de la surface de la table, ni des carottes qui ont la même couleur. Ceci peut poser problème pour reconnaître des objets, si les détails et les ombres ne sont pas représentées par une classe de pixels, et que les couleurs des objets sont assez similaires...

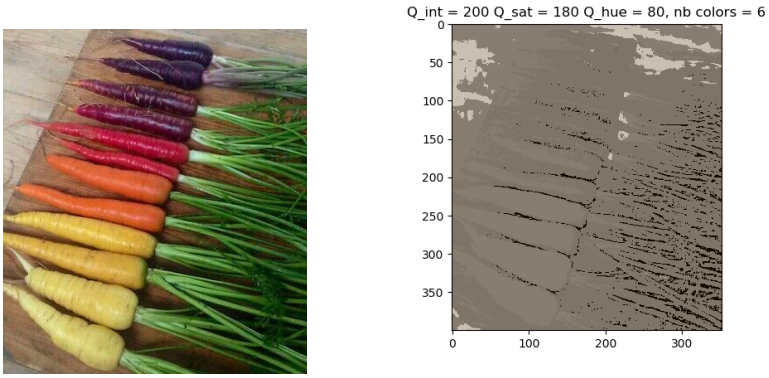

FIG.9 : À DROITE, IMAGE *carottes* ; À GAUCHE, RÉSULTAT DE LA FONCTION ACOPA

L'algorithme de k-means, qui réajuste les centres en prenant en compte la forme des clusters autour d'un mode, est donc nécessaire pour terminer le calcul de notre palette de couleurs.

3.2 Calcul des couleurs représentatives après k-means, et images obtenues à la fin

Dans notre fonction ACoPa, nous avons enregistré les couleurs qui seront finalement représentées sur l'image, et leur nombre. On applique ensuite l'algorithme de k-means à l'image avec ces centres initiaux. En récupérant les nouveaux centres des clusters, on applique la fonction kmeans.predict pour retrouver dans quelle classe est finalement chaque pixel, et on lui affecte la valeur du centre de la classe.

Cette méthode retire le choix difficile du paramètre k et des centres de k-means, car tous deux sont déterminés de manière unique pour chaque image, en se basant sur son histogramme. Les résultats après k-means sont grandement améliorés, pour *carottes* et *ladybug* que nous avons vues plus haut par exemple.

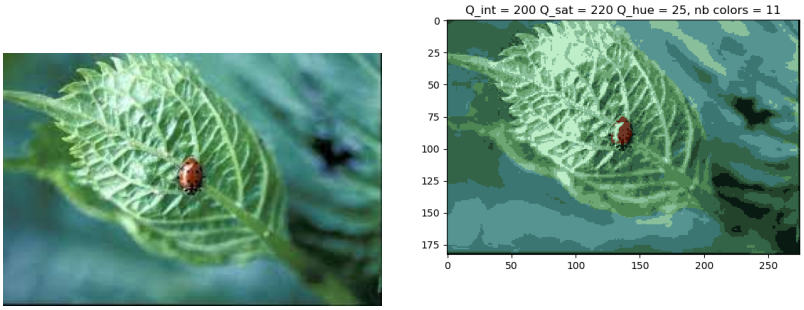

FIG.10 : À GAUCHE, IMAGE *ladybug* ; À DROITE, RÉSULTAT FINAL

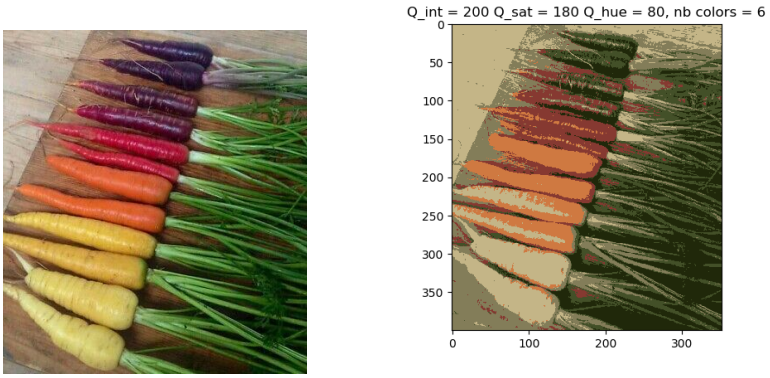

FIG.11 : À GAUCHE, IMAGE *carottes* ; À DROITE, RÉSULTAT FINAL

Nous devons cependant souvent ajuster nos paramètres de quantification en fonction des images, même si nous avons remarqué qu'une valeur de I autour de 255, de H en dessous de 100 et de S autour de 200 suffisaient dans la plupart des cas.

4 Test sur des images présentant des difficultés particulières

Nous avons essayé de sélectionner un certain nombre d'images qui pourraient présenter des difficultés particulières.

4.1 Image avec un contraste faible

Afin de voir comment notre algorithme réagissait à une image avec faible contraste, nous avons décidé de travailler avec cette image :

FIG.12 : IMAGE CHOISIE POUR LE TEST DE CONTRASTE FAIBLE.

La qualité de la palette dépend de finesse de la quantification de la saturation, car il faut essayer d'être le plus fin possible au vu du contraste faible. Avec Q_sat de 180 puis 250, on obtient les résultats suivants :

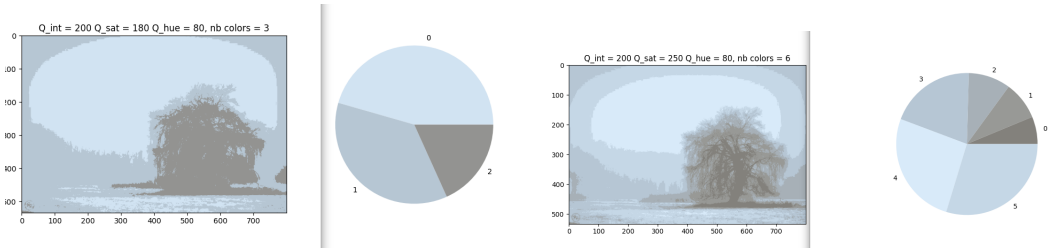

FIG.13 : Q_SAT = 180 À GAUCHE, Q_SAT = 250 À DROITE

Le résultat est plutôt concluant, dans la mesure où avec les 6 couleurs on obtient une précision satisfaisante sur les détails tels que la ramure de l'arbre par exemple. La géométrie de l'image est reconnaissable.

4.2 Faible luminosité

Dans cet exemple également, il s'est agi de jouer sur la précision de la quantification, ici sur I, qui caractérise l'intensité moyenne de l'image. L'image utilisée pour cela est une version assombrie de Lena.

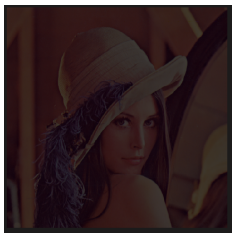

FIG.14 : IMAGE CHOISIE POUR LE TEST DE LUMINOSITÉ FAIBLE.

On remarque que même en augmentant beaucoup le critère de quantification sur l'intensité, on n'obtient pas une augmentation significative du nombre de couleurs, néanmoins les couleurs sélectionnées permettent une très bonne représentation de l'image d'origine.

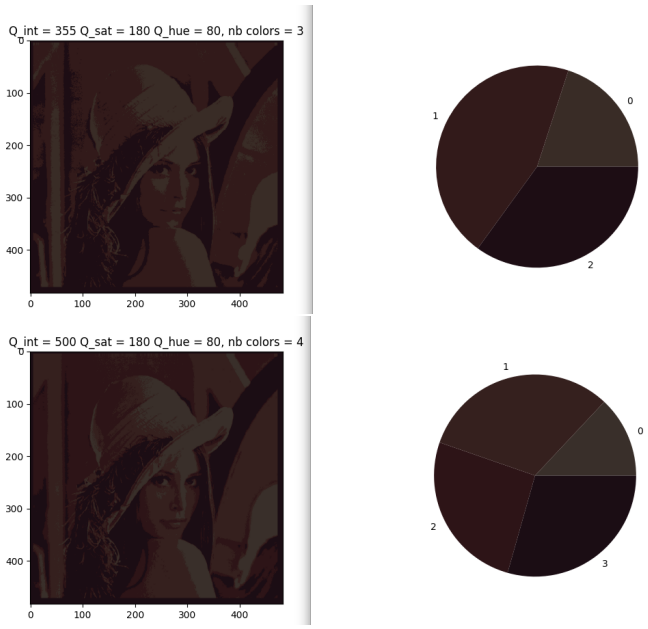

FIG.15 : Q_INT = 355 EN HAUT, Q_INT = 500 EN BAS

4.3 Image multicolore

L'objectif était de voir si l'algorithme avait des difficultés à représenter une image avec beaucoup de couleurs différentes.

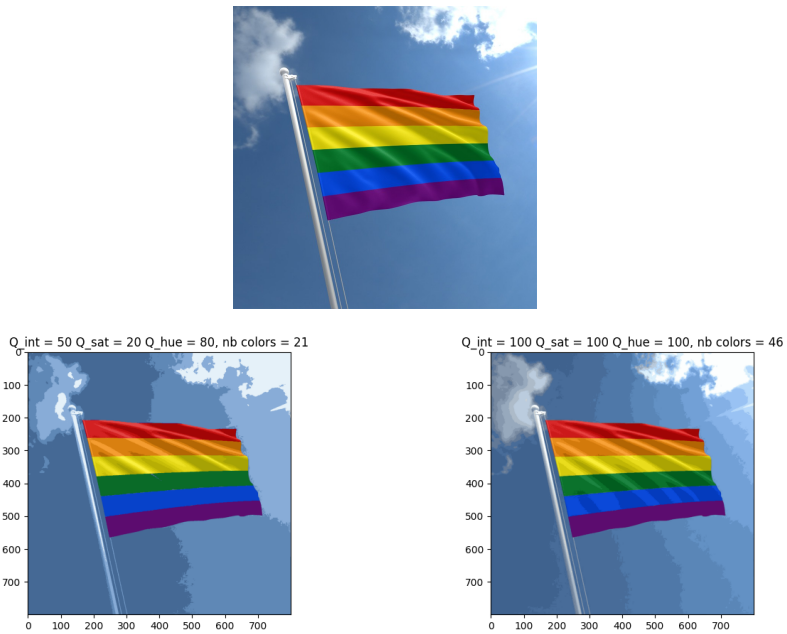

FIG.16 : IMAGE ORIGINALE EN HAUT, 21 COULEURS À GAUCHE, 46 À DROITE

Il est intéressant d'observer le fait que plus l'on augmente la précision sur S et I, plus l'on va pouvoir récupérer les détails traduisant les ondulations du drapeau sur les bandes de couleur inférieures du drapeau. Cela peut s'expliquer du fait que sur l'image originale, la différence d'intensité entre les parties éclairées et sombres de ondulations est plus importantes sur les couleurs du haut du drapeau. Ainsi, à facteur de quantification égal, la saturation et l'intensité sera mieux segmentée pour les teintes chaudes que pour les teintes froides.

Nous avons également utilisé une image avec une quantité beaucoup plus importante de couleurs différentes. Le résultat est également satisfaisant.

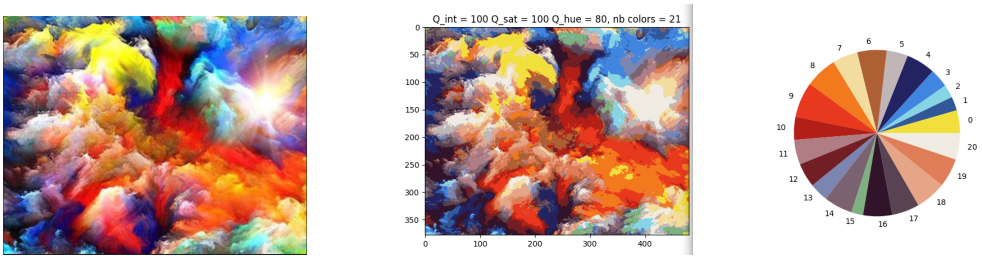

FIG.17 : IMAGE ORIGINALE À GAUCHE, VERSION AVEC 21 COULEURS À DROITE.

4.4 Dégradé

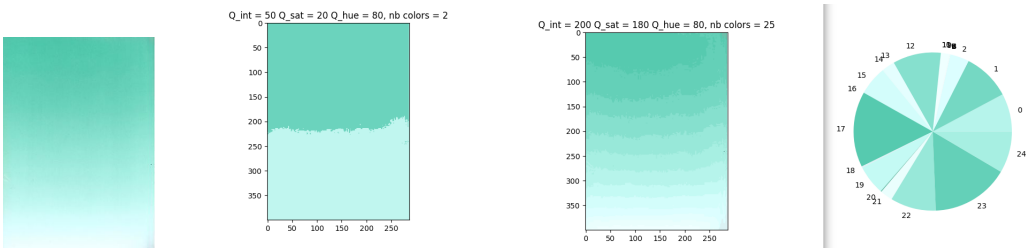

FIG.18 : DE GAUCHE À DROITE : IMAGE ORIGINALE, VERSION 2 COULEURS ET VERSION 25 COULEURS

Sans grande surprise, augmenter la précision sur S et sur I permet d'obtenir une représentation beaucoup plus fidèle de l'image de base. Ce type d'image permet notamment de démontrer en quoi l'utilisation de l'espace HSI est plus pertinent que l'espace BGR ou RGB. En effet, la teinte globale de l'image est similaire et ce qui va caractériser les différents pixels va plutôt être leur saturation ou l'intensité. Il paraît donc particulièrement adapté de chercher à séparer les pixels selon ces critères-là, qui sont beaucoup plus difficiles à mesurer dans l'espace RGB par exemple.

4.5 Texture

Cette méthode est particulièrement intéressante pour conserver les informations d'une image avec une certaine texture (motif répétitif, que l'on cherche à représenter avec assez peu d'informations). On voit en effet que même pour une image qui semble complexe et dont les couleurs sont difficiles à identifier pour l'oeil humain, assez peu de couleurs suffisent pour recréer la texture.

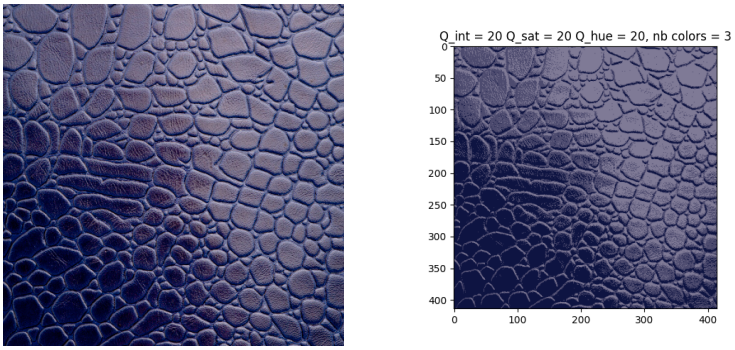

Fig.19 : de gauche à droite : image originale, version 2 couleurs et version 25 couleurs

5 Critiques et pistes d'approfondissement

L'algorithme nécessite à l'heure actuelle que l'on joue avec les valeurs de quantification de H, S et I, donc que l'on fasse un pré-traitement manuel des images, pour estimer la finesse nécessaire pour une bonne quantification. Cela est moins contraignant que de devoir extrapoler le nombre de couleurs nécessaires pour représenter l'image, mais n'est pas totalement satisfaisant si l'on souhaite n'avoir qu'à donner une image en entrée, sans avoir à fournir de valeurs ou d'informations supplémentaires.

Une piste intéressant d'approfondissement de ce projet pourrait être d'automatiser cet ajustement. Analyser les données statistiques sur H, S, I pourrait permettre de déduire des caractéristiques de l'image (par exemple un écart-type et une moyenne faibles sur I permettrait de caractériser les images avec une faible luminosité). Il serait alors ensuite possible d'adapter la quantification afin de l'affiner sur les données nécessitant une précision plus grande.

Ensuite, la conversion vers HSI est relativement longue, car de l'ordre de la taille de l'image. Il est donc coûteux de chercher à traiter des images de taille conséquente. Une piste intéressante pourrait être de voir si cette méthode donnerait des résultats probants dans un espace plus habituel, comme l'espace HSV par exemple. Un des inconvénients de l'espace HSI est le fait qu'il n'existe pas de bibiothèque Python courante de gestion d'images, comme opencv, qui permettent de faire la conversion. Les calculs de conversion sont donc codés en dur, que ce soit pour aller de BGR vers HSI (la méthode imread de opencv chargeant les images en BGR) ou de HSI vers RGB. Opencv possède en revanche des fonctions permettant une conversion de BGR vers HSV par exemple.