



École nationale
de la statistique
et de l'analyse
de l'information



ÉCOLE NATIONALE DE LA STATISTIQUE
ET DE L'ANALYSE DE L'INFORMATION

La segmentation en superpixel

STAGE 2A

Etudiant:

Raphaël GERVILLIE

Encadrant:

Myriam VIMOND

Abstract

La segmentation en superpixel est devenue en quelques années une méthode de pré-traitement de plus en plus utilisée dans les applications de vision par ordinateur. Cela a conduit au développement de nombreux algorithmes de segmentations en superpixel, ayant tous leurs avantages et leurs inconvénients. Durant ce mémoire, nous verrons en détail deux méthodes de segmentations en superpixels, le SLIC et l'algorithme de Felzenszwalb et Huttenlocher. Nous discuterons également d'une méthodologie permettant de mesurer la qualité des superpixels créés par ces algorithmes. Cette méthodologie sera ensuite utilisée pour effectuer un étude comparative de ces deux algorithmes et permettra alors de dégager leurs points forts et leurs points faibles. Mais également d'aider à comprendre comment choisir un algorithme en fonction des propriétés désirées sur les superpixels.

Contents

1	Introduction	2
2	Données, enjeux et métriques.	4
2.1	Représentation numérique d'une image	4
2.2	Les enjeux de la segmentation	5
2.3	Evaluer la performance des superpixels.	5
3	Le K-means	8
3.1	K -means et Superpixels	8
3.2	Consistance des K -means	9
3.3	Expériences et interprétations	9
4	Simple Linear Iterative Clustering	11
4.1	Le SLIC comme adaptation des K-means.	11
4.2	L'algorithme SLIC complet	12
4.3	Discussion sur le SLIC	12
5	Algorithme de Felzenszwalb et Huttenlocher	14
5.1	Les graphes non orientés, connexes	14
5.2	Minimum Spanning Trees	14
5.3	Critère pour comparer deux régions	16
5.4	Algorithme complet.	17
6	Comparaison de méthodes	18
6.1	L'importance du paramétrage dans SLIC.	18
6.2	L'importance du paramétrage dans FH.	20
6.3	Comparaison de la segmentation par graphe et de SLIC.	20
7	Conclusion et perspectives	22
A	Annexes	23
A.1	Les espaces de couleur RGB, LAB	23
A.2	Exemple de vérité terrain	23
A.3	Exemple d'enveloppe convexe	24
A.4	Exemple de Minimum Spanning Trees	24

Chapter 1

Introduction

Une segmentation en superpixels consiste à découper une image en groupements de pixels, appelés superpixels. Ces derniers doivent être assez proches en taille et en forme, coller aux frontières de l'objet, ou encore avoir des couleurs homogènes Wang et al. [2017].

La segmentation en superpixels sert généralement comme étape de pré-traitement d'images pour des tâches de vision par ordinateur. On l'utilise par exemple pour la reconnaissance de formes Yang et al. [2014] ou encore pour le suivi visuel Giordano et al. [2015]. Travailler sur un groupement de pixels plutôt que des pixels présente de nombreux avantages. D'une part, la segmentation en superpixels contribue à une représentation plus simplifiée de l'image et donc à diminuer le temps de calcul associé à leur traitement. D'autre part, ils fournissent plus d'information spatiales que les pixels et permettent ainsi de mieux mesurer certaines caractéristiques régionales Hu et al. [2015].

Pour créer ces superpixels, de nombreux algorithmes ont vu le jour dans la littérature (SEEDS, GMM, ERS, NCut, ...). Ces méthodes peuvent être séparées en deux classes principales Wang et al. [2017] : les méthodes basées sur les graphes et les méthodes basées sur le gradient. Afin d'avoir une vision globale de la segmentation en superpixels, ce mémoire présente un algorithme de gradient le SLIC, puis l'algorithme de segmentation en superpixels de Felzenszwalb et Huttenlocher (FH) basé sur les méthodes de graphes. Ce sont deux algorithmes qui ont pour objectif le partitionnement des données mais qui vont l'effectuer d'une façon très différente, on peut déjà constater cette différence dans la forme que prennent les superpixels (voir figure 1.1).

Pour pouvoir comprendre quels sont les avantages et les inconvénients de ces algorithmes, il est important d'étudier les propriétés de ces derniers. Ce rapport vise donc à proposer une méthodologie complète permettant d'évaluer la qualité d'une segmentation. Puis de l'utiliser, afin de réaliser une étude comparative entre le SLIC et le FH. Cette méthodologie aura également pour objectif de pouvoir être appliquée à d'autres algorithmes de segmentation et s'inscrit donc dans le besoin croissant de mieux comprendre ces méthodes.

Ce mémoire vise donc à présenter des algorithmes de segmentation en superpixels. Mais également à proposer une méthodologie afin de découvrir leurs points forts et points faibles. L'ensemble du travail durant ce stage a été réalisé sur le langage python, les principaux résultats sont disponibles à cette adresse <https://github.com/RaphaelGervillie/Superpixel>.

Ce document est organisé de la manière suivante. Le chapitre 2 décrit les données utilisées, les propriétés des algorithmes de segmentation en superpixels et introduit la méthode d'évaluation de ces propriétés. Un rappel sur les K -means sera fait dans le chapitre 3. Le chapitre 4 permettra d'introduire le premier algorithme de segmentation SLIC et le chapitre 5 l'algorithme de Felzenwalb et Huttenlocher. La comparaison des méthodes sera faite dans le chapitre 6. Enfin, nous conclurons dans le chapitre 7.

Laboratoire d'accueil : CREST

Le CREST (Centre de Recherche en Economie et Statistiques) est un centre de recherche qui regroupe des professeurs de l'ENSAE, l'ENSAI et du département d'économie de l'Ecole Polytechnique. Ce centre comprend quatre domaines principaux, l'économie, les statistiques, la finance-assurance et la sociologie.

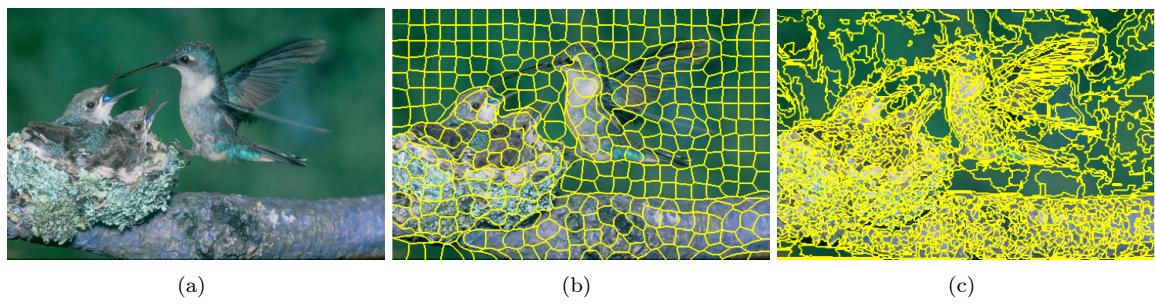


Figure 1.1: (a) Image d'origine. (b) Représentation de la segmentation en superpixel avec l'algorithme SLIC pour $k=350$ et $M=15$. (c) Représentation de la segmentation en superpixel avec l'algorithme de Felsenwald et Huttenlocher pour $z=10$.

Chapter 2

Données, enjeux et métriques.

2.1 Représentation numérique d'une image.

Une image digitale I se divise en N lignes et M colonnes, l'intersection d'une ligne et d'une colonne se nomme alors le pixel. Ainsi on note par P_{xy} le pixel aux coordonnées (x, y) avec $\{x = 0, 1, \dots, M - 1\}$ et $\{y = 0, 1, \dots, N - 1\}$. Lorsque l'on travaille sur des images en noir et blanc, chaque pixel est défini par une variation de la luminosité entre le blanc et le noir. Dans ce cas, on note $P_{xy}(b)$ le pixel (x, y) de luminosité b , Young et al. [1998]. Enfin, pour une image couleur, on ne parle plus de variation de l'intensité entre le blanc et le noir, mais de variation de l'intensité des trois couleurs rouge, vert et bleu. On passe donc d'une seule à trois dimensions, d'où les trois matrices représentées sur la figure 2.1. Par exemple, la matrice en premier plan contient les intensités de rouge des pixels. Cet espace de couleurs s'appelle le *RGB* pour Red (Rouge), Green (vert) et Blue (bleu). Les intensités sont ici standardisées entre 0 et 1, mais sont généralement comprises entre 0 et 255. On note alors un pixel de coordonnées (x, y) et d'intensité (r, g, b) avec $\{r = 0, 1, \dots, 255\}$, $\{g = 0, 1, \dots, 255\}$ et $\{b = 0, 1, \dots, 255\}$, $P_{xy}(r, g, b)$.

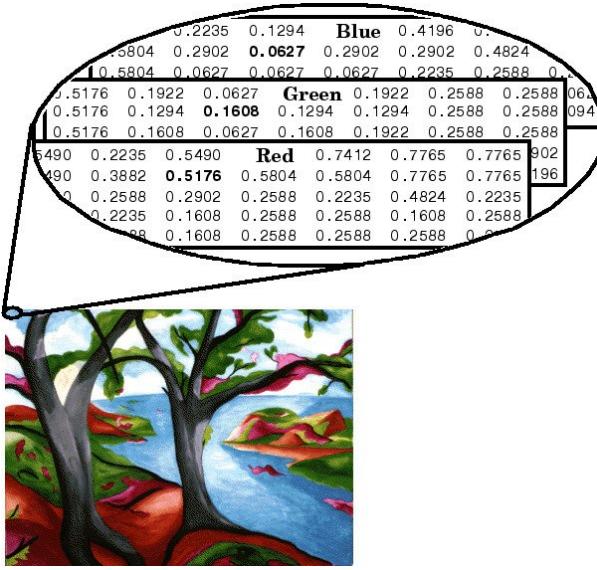


Figure 2.1: Représentation d'une image couleur dans l'espace couleur RGB. © 1994-2020 The MathWorks, Inc.

Bien que l'espace couleur *RGB* soit le plus populaire, lors d'une analyse de couleur il est courant d'utiliser l'espace *LAB**. En effet, "la distance euclidienne entre deux points de couleur dans l'espace colorimétrique *LAB** correspond à la différence de perception entre les deux couleurs par le système de vision humaine". Acharya and Ray [2005] Il est démontré par exemple que l'on obtient de meilleures performances,

en l'utilisant sur les applications de regroupement de couleurs. Ainsi un grand nombre d'algorithmes de segmentation en superpixel utilisent l'espace *LAB** et non *RGB*. Il est néamoins possible de passer d'un espace à un autre (voir annexe A.1). Comme précédemment on note un pixel de coordonnées (x, y) et d'espace couleur $(l, a, b*)$, $P_{xy}(l, a, b*)$.

2.2 Les enjeux de la segmentation

Cette représentation numérique dans notre cas, a pour but d'utiliser des algorithmes de segmentation en superpixels. L'objectif de telles méthodes est de découper une image en groupements de pixels, appelés superpixels. Ils doivent être assez proches en taille et en forme, mais aussi coller aux frontières de l'objet, ou encore avoir des couleurs homogènes. Cette différence sur la taille des superpixels permet de distinguer cette méthode des méthodes de segmentation des formes. On note S_k le superpixel k , avec $k = 1, \dots, K$, K étant le nombre total de superpixels créés. Dans la littérature Wang et al. [2017], Ban et al. [2018], leurs caractéristiques sont bien connues, elles sont résumées ci-dessous.

- **La précision** est la caractéristique du superpixel à coller aux frontières des objets dans une image. Le non-respect de cette propriété peut engendrer des conséquences graves sur les étapes de post-traitement, comme par exemple pour la reconnaissance de formes.
- **La régularité des formes** d'un superpixels permet de mener une meilleure analyse de l'image, et donc de réaliser une meilleure performance sur les applications de reconnaissance de formes. Giraud et al. [2017]
- **L'homogénéité des couleurs** du superpixel exige de la variation d'intensité d'être la plus faible possible. Autrement dit, les pixels au sein d'un même superpixel doivent être très similaires en terme d'intensité de couleur.
- **L'efficacité computationnelle** pour la mise en application sur des problématiques réelles.

La segmentation en superpixels étant une étape de pré-traitement, les algorithmes doivent respecter ces propriétés pour faciliter les étapes suivantes. Ces méthodes de segmentation deviennent de plus en plus importantes avec l'utilisation exponentielle des applications dites de vision par ordinateur. En effet la vision par ordinateur a pour objectif de faire comprendre une image ou une vidéo à un ordinateur. Or quand le nombre d'images, le temps d'une vidéo ou encore leur qualité augmentent, les temps de calcul deviennent de plus en plus longs. Pour répondre à cette problématique, la segmentation en superpixels est devenue une étape de pré-traitement fondamentale. On la retrouve par exemple :

- Dans le tracking Yang et al. [2014], dans leur publication, les auteurs proposent une nouvelle méthode de reconnaissance d'objet, se basant sur la segmentation en superpixels. Ils montrent que cette dernière permet d'obtenir des repères visuels et d'améliorer la capacité du tracker d'objets à reconnaître l'objet de premier et d'arrière-plan.
- Dans ce deuxième article Giordano et al. [2015] les auteurs utilisent des vidéos et cherchent à en reconnaître les objets. D'après eux : « La modélisation basée sur les superpixels améliore non seulement les performances de segmentation mais augmente également l'efficacité, en terme de mémoire requise et de temps de calcul ».

2.3 Evaluer la performance des superpixels.

Comme évoqué précédemment, un superpixel doit respecter un ensemble de conditions pour être exploité ensuite comme étape de pré-processing, avant les applications de computer vision. Afin d'évaluer ces propriétés, nous introduisons plusieurs métriques.

L'homogénéité des couleurs [Giraud et al., 2017]. Cette propriété donne le niveau d'homogénéité des couleurs d'un superpixel et peut être mesurée par la variation expliquée sur chaque canal de couleur $C \in \{R, G, B\}$,

$$EV_c = \frac{\sum_{k=1}^K |S_k|(\mu_C(S_k) - \mu_C(I))^2}{\sum_{i=1}^N M(P_i(C) - \mu_C(I))^2}, \quad (2.1)$$

où $\mu_C(S_k)$ est l'intensité moyenne d'un superpixel S_k sur le canal C , $|S_k|$ le nombre de pixels associés à S_k , $\mu_C(I)$ l'intensité de moyenne de l'image I sur le canal de couleur C et $P_i(C)$ le pixel des coordonnées i du canal de couleur C avec $\{i = 1 \dots N * M\}$. Les mesures de l'intensité moyenne et de la variation de l'intensité sont représentées sur la figure 2.2. Il est important de noter qu'à une forte valeur de l' EV correspond unclustering qui respecte bien l'homogénéité des couleurs. En effet si la variance à l'intérieur d'un superpixel augmente alors le critère de l' EV diminue.

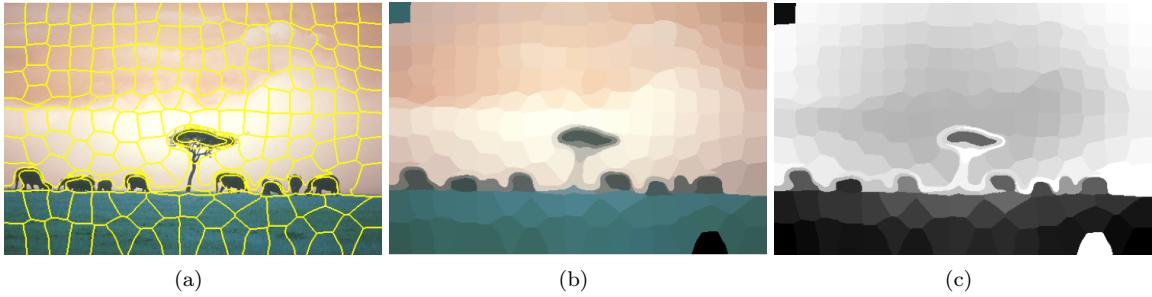


Figure 2.2: (a) Représentation de la segmentation en superpixel d'une image couleur avec le SLIC pour $K=200$ et $m=5$. (b) Représentation de l'intensité moyenne de chaque superpixel $\mu_C(S_k)$. (c) Représentation de la variation de l'intensité de Rouge sur chaque superpixels $\frac{1}{|S_k|-1}(\mu_R(S_k) - \mu_R(I))^2$.

L'adhérence à la frontière de l'image [Levinshtein et al., 2009]. Pour calculer le niveau de proximité des superpixels avec les bordures de l'image, on introduit la notion de vérité terrain. Obtenue dans le dataset Berkley Martin et al. [2001] une vérité terrain correspond à des segmentations déjà mesurées. On notera Gt la vérité terrain t avec $t = 1, \dots, T$, T étant le nombre total de vérités terrain. Ces dernières sont utilisées dans cette partie pour mesurer l'adhérence de nos superpixels aux frontières de l'image (voir l'annexe A.1 pour des exemples de vérités terrains). Une vérité terrain divise un superpixel S_k dans une partie S_{in} et S_{out} . Ceci est illustré sur la figure 2.3. On a dans cet exemple $G_{out} = \frac{|A_{out}| + |B_{out}| + |C_{out}|}{|GroundTruthSegment|}$.

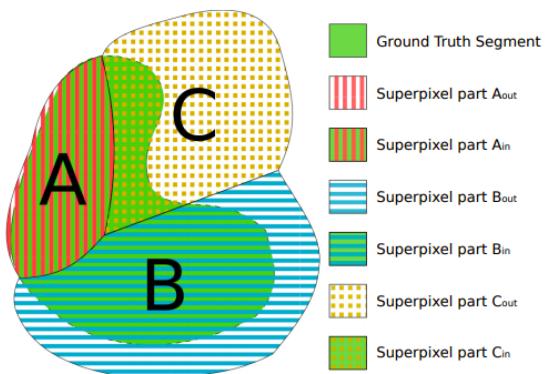


Figure 2.3: Représentation de l'erreur de segmentation, ici le ground truth est chevauché par trois superpixels (A, B, C), Neubert and Protzel [2012]

Il existe un grand nombre de métriques pour mesurer l'adhérence à la frontière, la plus populaire Giraud et al. [2017] donnant la somme pour chaque superpixel de la partie G_{out} .

$$UndersegmentationError1 = \sum_{t=1}^T \frac{\sum_{S:S \cap G_t \neq \emptyset} G_{out}}{|G_t|} \quad (2.2)$$

Comme évoqué dans Neubert and Protzel [2012] l'inconvénient de cette mesure réside dans la pénalité conséquente imposée à un superpixel qui traverse très légèrement une vérité terrain, c'est-à-dire qui se compose d'une grande partie S_{out} et d'une très petite S_{in} . L'auteur propose de corriger cette limite dans une deuxième version de la métrique.

$$UndersegmentationError2 = \frac{1}{N} \sum_{t=1}^T \left(\sum_{S:S \cap G_t \neq \emptyset} \min(G_{out}, G_{in}) \right) \quad (2.3)$$

Cette deuxième définition est moins sensible au chevauchement des superpixels avec les vérités terrain. On remarque également que chaque chevauchement des superpixels avec la vérité terrain engendre une hausse de cette métrique. Ainsi plus la métrique "UndersegmentationError2" est faible plus les superpixels respectent les frontières de l'image.

La compacité [Schick et al., 2012]. Le problème de mesure de la compacité, qui consiste à « trouver la forme bidimensionnelle qui a la plus grande surface possible pour une longueur limite donnée » a déjà été résolu par les mathématiques. La solution étant le cercle, car sa forme est la plus compacte, on se base sur ce résultat pour construire la métrique. On note A_s l'aire et L_s le périmètre d'un superpixel. La mesure de compacité nous est alors donnée par :

$$Q_s = \frac{A_s}{A_c} = \frac{4\pi A_s}{L_s^2} \quad (2.4)$$

Cette métrique est très performante pour évaluer la forme d'un superpixel individuellement, mais échoue pour donner un score sur l'ensemble de l'image comme le suggère Schick et al. [2012]. Pour évaluer la régularité des superpixels dans leur ensemble, il introduit $CC(S_k) = Perim(S_k)/|S_k|$ et $CC(H_k) = Perim(H_k)/|H_k|$ avec $Perim(S_k)$ le périmètre du superpixel k et H_k l'enveloppe convexe du superpixel k . Un exemple d'une enveloppe convexe d'un superpixel est disponible en annexe A.2. On obtient ainsi le critère de régularité

$$CR(S_k) = \frac{CC(H_k)}{CC(S_k)}. \quad (2.5)$$

"Puisque l'enveloppe convexe H_k contient entièrement S_k et a un périmètre inférieur, la mesure CR est comprise entre 0 et 1, et est maximale pour les formes convexes telles que les carrés, les cercles ou les hexagones." Schick et al. [2012]. Néanmoins cette mesure reste insuffisante car elle est maximale pour des ellipses ou des lignes. Il faut donc prendre en compte la répartition équilibrée des pixels dans la forme

$$V_{xy}(S) = \frac{\min(\sigma_x, \sigma_y)}{\max(\sigma_x, \sigma_y)}. \quad (2.6)$$

Cette équation prend en compte cet aspect, on note σ_x, σ_y la déviation standard de la position du superpixel en x et en y . Ainsi on obtient $V_{xy}(S) = 1$ uniquement si $\sigma_x = \sigma_y$ c'est-à-dire si la répartition spatiale des pixels est équilibrée. Dans le cas extrême où le superpixel correspond à une ligne, on a alors $V_{xy}(S) = 0$. On obtient finalement notre métrique pour mesurer la régularité des formes,

$$SRC(S) = \sum_{S_k} \frac{|S_k|}{|I|} CR(S_k) V_{xy}(S_k). \quad (2.7)$$

Cette métrique sera performante pour des formes carrées, des cercles ou des hexagones, on cherche donc à maximiser ce critère.

Chapter 3

Le K-means

Ce chapitre propose un rappel sur l'algorithme des K -means puisque son fonctionnement est repris dans l'algorithme de segmentation en superpixel SLIC.

3.1 K -means et Superpixels

Le K -means est un algorithme de clustering, c'est-à-dire qu'il a pour but de regrouper les données dans des grappes homogènes. « Intuitivement on cherche donc à créer des grappes dont les éléments sont très proches les uns des autres comparés aux éléments à l'extérieur de ces grappes. » Bishop [2006]

Supposons que nous avons N observations d'une variable aléatoire à D dimensions, $\{X_1, \dots, X_N\}$ que nous cherchons à répartir en K grappes avec K le bon nombre de grappes supposé connu. On regroupe les éléments en grappes en fonction de la distance de chaque observation au K centroïdes $u_1 \dots u_K$. On note $r_{nk} \in \{0, 1\}$, $r_{nk} = 1$ si l'observation X_n est assignée au cluster K et inversement. On introduit maintenant la mesure de distorsion du K -means :

$$J = \sum_{n=1}^N \sum_{k=1}^K r_{nk} \|x_n - u_k\|^2. \quad (3.1)$$

Cette mesure correspond au carré de la distance entre chaque observation et le cluster où il est associé. Plus J sera faible plus la qualité du clustering sera bonne. Il s'agit donc de trouver les valeurs de r_{nk} et de u_k permettant de minimiser cette mesure. Le K -means réalise cette minimisation de manière itérative, où chaque itération suit deux grandes étapes :

- La première cherche à minimiser J en fonction de r_{nk} , avec u_k fixé. Par hypothèse, les données sont indépendantes, il est donc possible d'optimiser séparément. On aura donc $r_{nk} = 1$ si pour toutes les valeurs K l'observation est minimale en $\|x_n - u_k\|^2$. C'est-à-dire que parmi l'ensemble des centroïdes, l'observation X_n est la plus proche du centroïde K . Cette étape d'optimisation est très intuitive car il s'agit simplement d'attribuer à chaque observation le centroïde dont elle est le plus proche.
- La seconde optimisation va cette fois minimiser J en fonction de u_k , avec r_{nk} fixé. En dérivant par u_k et fixant comme égale à 0 on obtient :

$$2 \sum_{n=1}^N r_{nk} (x_n - u_k) = 0, \quad (3.2)$$

ce qui nous donne

$$u_k = \frac{\sum_{n=1}^N r_{nk} X_n}{\sum_{n=1}^N r_{nk}}. \quad (3.3)$$

On voit finalement apparaître dans cette formule que u_k est bien égale à la moyenne de toutes les observations assignées au cluster k . Les deux étapes décrites ci-dessus sont répétées jusqu'à ce qu'il n'y ait plus de changement dans l'assignement des données, ou bien jusqu'à ce qu'un certain critère d'arrêt soit dépassé. Ainsi chaque étape d'optimisation réduit la valeur de la fonction objective J , la convergence de l'algorithme est donc assurée. Cependant, il peut converger vers un minimum local plutôt que global.

3.2 Consistence des K -means

La preuve de la consistence des K -means est démontrée dans Pollard [1981]. Il formule le problème des K -means comme un problème de minimisation de la fonction

$$W(U, P_n) := \int \min_{u_k \in U} \|x - u_k\|^2 P_n(dx) \quad (3.4)$$

avec P_n la distribution empirique. En minimisant ce critère on divise alors nos données en K clusters. Afin d'obtenir des clusters optimaux, deux hypothèses doivent être respectées. D'abord, $\{X_1, \dots, X_N\}$ doivent être des variables indépendantes d'une certaine distribution P . Mais surtout U les centroïdes initialement choisis, doivent être pris de façon optimale c'est à dire qu'ils doivent être du bon nombre K . Sous ces conditions, on a une convergence asymptotique des centroïdes initiaux U vers les centroïdes optimaux qu'on notera U^* , c'est-à-dire qu'on va trouver les centroïdes qui permettent de minimiser $W(A, P)$, P étant la vraie distribution de nos données $U \rightarrow U^*$. Plus formellement On voit que la convergence des K -means ne peut donc être assurée qu'en connaissant à priori le nombre de clusters K , mais également en ayant suffisamment de données car la convergence est asymptotique.

3.3 Experiences et interprétations

Comme expliqué précédemment, la particularité des K -means vient de son aspect itératif, qui consiste à recalculer les centres des clusters à chaque étape. Sur la figure 3.1 on y voit l'implémentation du K -means dans un problème à deux dimensions où les observations (X_i) sont les coordonnées spatiales, avec le nombre de classes k à priori connues. On voit alors comment l'algorithme recalcule la moyenne pour modifier les centroïdes et s'arrête quand leur position cesse d'évoluer.

Nous venons de voir comment le K -means converge après plusieurs itérations. Nous allons maintenant voir qu'il est possible d'utiliser les K -means pour le faire fonctionner sur des images. Il s'agit alors de lui donner en donnée d'entrée l'intensité de couleur des pixels. Dans cette situation le K -means n'est plus calculé sur l'espace XY mais sur un espace couleur LAB^* en dimension 3. Cette méthode peut être utilisée de manière basique pour la reconnaissance de formes comme on peut le voir dans la figure 3.2.

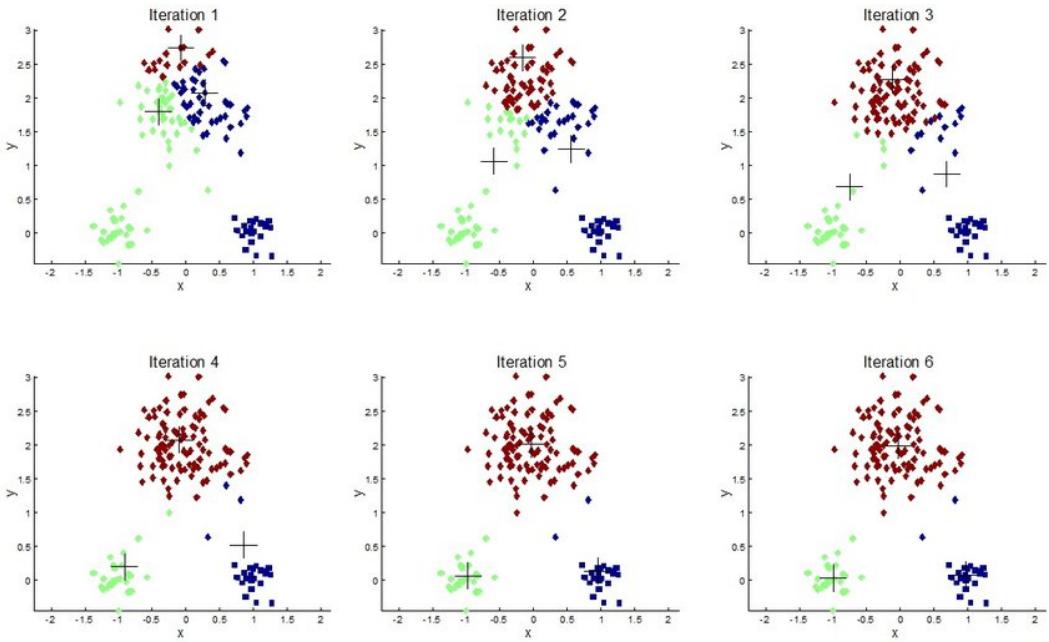


Figure 3.1: La convergence du K-means après chaque itération. Bishop [2006]

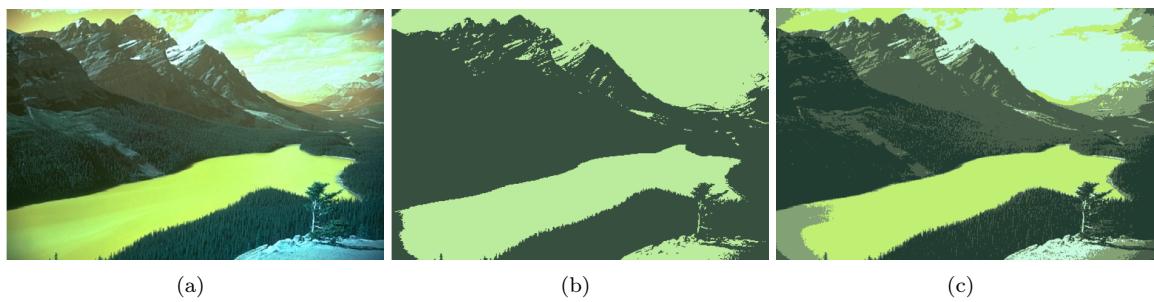


Figure 3.2: Illustration du K -means utilisé pour la segmentation d'objet. Le K -means a été utilisé sur l'espace de couleur LAB^* donc sans prendre en compte les positions spatiales des pixels. (a) Image originelle (b) Représentation des clusters obtenus avec $k=2$ après 10 itérations de l'algorithme. On voit que l'image n'est composée que de deux couleurs. (c) Représentation des clusters obtenus avec $k=5$ après 10 itérations de l'algorithme.

Chapter 4

Simple Linear Iterative Clustering

Comme nous l'avons vu, les K -means peuvent être utilisés dans des applications de reconnaissance de formes en effectuant le clustering sur un espace de couleurs. Nous allons voir que l'algorithme *SLIC* reprend la structure des K -means et l'adapte afin de générer cette fois une segmentation en superpixels.

4.1 Le SLIC comme adaptation des K-means.

Pour bien comprendre SLIC Achanta et al. [2010] commençons par définir les notations importantes de cet algorithme. En données d'entrée il est possible de spécifier deux principaux paramètres. Le nombre K de superpixels que l'on souhaite générer, qui correspond au même K que dans les K -means, c'est à dire au nombre de grappes que l'on souhaite générer. On peut également spécifier le paramètre m , qui est un paramètre de compacité des superpixels. Plus m sera grand plus les superpixels seront compacts comme on peut l'observer dans la figure 4.1.

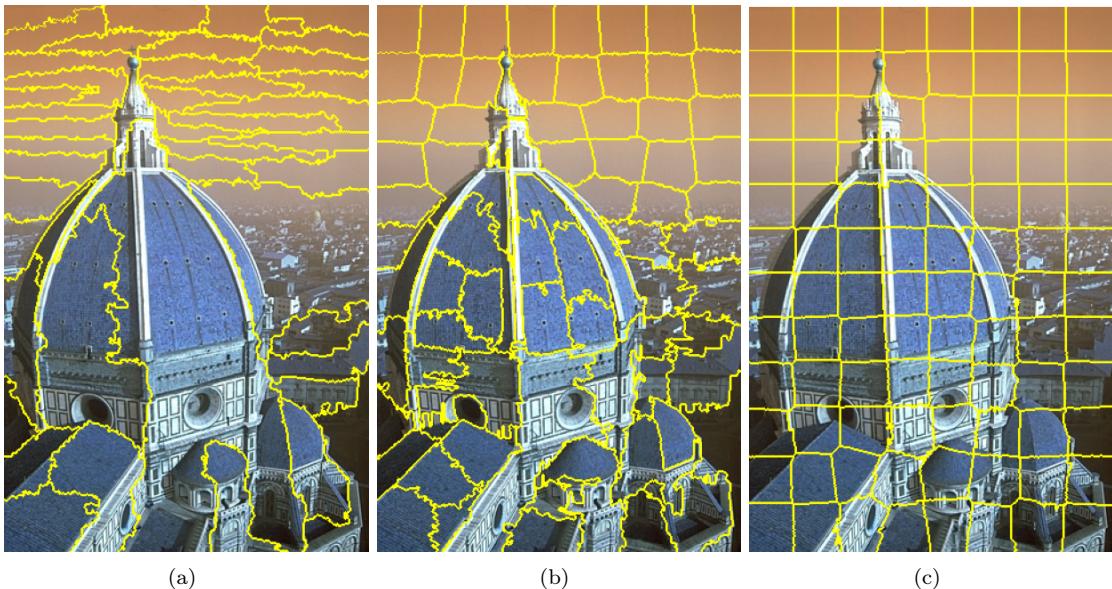


Figure 4.1: Illustration de l'impact du paramètre de compacité m sur la forme des superpixels. (a) $m=1$, $K=100$. (b) $m=10$, $k=100$. (c) $m=100$, $k=100$.

La taille de chaque superpixel sera donc environ de $\frac{N}{K}$ puisqu'on cherche à créer des superpixels homogènes.

Enfin nos superpixels seront éloignés d'une distance les uns des autres de

$$S = \sqrt{\frac{N}{K}}. \quad (4.1)$$

Comme évoqué précédemment, l'algorithme SLIC est très proche du K -means. La principale différence vient du calcul des distances entre chaque point. En effet le K -means peut calculer cette distance de manière spatiale (voir : 3.1) ou alors sur un espace de couleur (voir : 3.2). Alors que l'algorithme SLIC va classifier des pixels en fonction de leur similitude en terme de distance et de couleur. Il n'est pas possible d'utiliser la distance euclidienne en \mathbb{R}^5 sans normalisation. Ainsi le calcul de la distance D_s correspond à la somme entre la distance LAB et la distance spatiale XY normalisée par l'intervalle entre chaque cluster. Le paramètre m sert de paramètre de compactité, en l'augmentant on augmente l'importance de la distance spatiale dans le calcul de D_s et donc on force nos superpixels à être compact.

$$D_{lab} = \sqrt{(l_k - l_i)^2 + (a_k - a_i)^2 + (b_k - b_i)^2} \quad (4.2)$$

$$D_{xy} = \sqrt{(x_k - x_i)^2 + (y_k - y_i)^2} \quad (4.3)$$

$$D_s = D_{lab} + \sqrt{\frac{m}{S}} D_{xy} \quad (4.4)$$

C'est cette distance D_s qui permet de distinguer l'algorithme de segmentation des formes K -means et l'algorithme SLIC de segmentation en superpixel. En effet en prenant en compte les dimensions spatiales et leur couleur, il génère des clusters avec des formes régulières.

4.2 L'algorithme SLIC complet

Je vais enfin vous commenter le déroulé général de l'algorithme SLIC comme présenté dans l'article Achanta et al. [2010]. On commence par initialiser les K clusters, qu'on espaces d'une distance S les uns des autres. Puis on déplace ces clusters en fonction du gradient minimal calculé au voisinage de ce point. L'auteur nous explique que "ceci est fait pour éviter de les placer sur un bord et pour réduire les chances de choisir un pixel bruyant".

$$G(P(x, y, l, a, b*)) = \|P(x+1, y, l, a, b*) - P(x-1, y, l, a, b*)\|^2 + \|P(x, y+1, l, a, b*) - P(x, y-1, l, a, b*)\|^2 \quad (4.5)$$

Après l'initialisation on affecte chaque pixels au centroïde K qui lui est le plus proche en utilisant la distance D_s . Attention, cette distance ne sera calculée que si le pixel est dans un voisinage de taille $2s * 2s$ autour du centroïde k . Cette précision est une autre différence avec l'algorithme des K -means. Elle est extrêmement importante car elle permet de réduire drastiquement les coûts computationnels de l'algorithme SLIC, en comparaison avec les K -means Achanta et al. [2010]. D'après l'auteur on passerait d'un complexité de $O(NKIt)$ pour le K -means à une complexité de $O(N)$ pour le SLIC, avec It le nombre d'itérations requis pour la convergence. En effet la distance pour chaque pixel étant calculée uniquement s'il se trouve dans un voisinage $2s * 2s$ d'un centroïde, cela revient à calculer la distance de chaque pixel avec 8 centroïdes. De plus le nombre d'itérations est constant dans cet algorithme.

Puis, quand tous les pixels seront répartis au sein des clusters, on va calculer une nouvelle fois nos centroïdes comme égaux à la valeur moyenne en $(x, y, l, a, b*)$ de tous les pixels appartenant au cluster. On répète cette étape jusqu'à ce que le nombre d'itérations fixé soit dépassé.

4.3 Discussion sur le SLIC

Bien que le SLIC reprenne des éléments du K -means, il s'en distingue de par ses objectifs : alors que le K -means cherche à déterminer les grappes optimales, le SLIC lui doit avant tout respecter les propriétés évoquées dans la section 2.2. Autrement dit : créer des superpixels collant à la frontière de l'objet, qui sont

homogène en terme de couleur. Mais surtout, des superpixels dont les formes sont assez similaires, mais aussi le SLIC doit être assez rapide pour pouvoir être utilisé sur des applications de vision par ordinateur. Ces deux dernières conditions sont rendues possibles respectivement par la définition de la distance D_s et par la restriction au voisinage $2s * 2s$ du calcul des distances entre pixels et centroides, ainsi que par la limitation du nombre d'itérations de l'algorithme.

Chapter 5

Algorithme de Felzenszwalb et Huttenlocher

L'algorithme SLIC est basé sur une méthode de *clustering* itérative. L'algorithme de Felzenwalb et Huttenlocher (FH) est lui basé sur la méthode des graphes. Un graphe correspond à une représentation mathématique d'un problème, il est toujours composé d'un ensemble de noeuds et d'arêtes. Dans le cas d'une image, chaque pixel va correspondre à un noeud et les arêtes seront les liens existant entre ces pixels.

5.1 Les graphes non orientés, connexes

On note un graphe $G = (V, E)$ avec V un ensemble de noeuds et E un ensemble d'arêtes entre des noeuds. A chaque arête entre deux noeuds v_i et v_j est attribué un poids $w(v_i, v_j)$.

Pour construire les graphes sur une image, on considère chaque pixel comme un noeud, puis on calcule les poids. Concernant la définition de ces derniers, il existe différentes méthodes permettant de capturer la ressemblance entre chaque pixel ou la dissimilarité. Felzenszwalb and Huttenlocher [2004]. Dans leur article d'origine, Felzenszwalb et Huttenlocher utilisent deux méthodes pour calculer les poids. La première correspond à la différence d'intensité en valeur absolue entre deux pixels, par exemple pour le poids entre un pixel en (x, y) et un pixel en $(x + 1, y)$ on obtient :

$$w_r(p_{x+1,y}(r), p_{x,y}(r)) = |p_{x+1,y}(r) - p_{x,y}(r)|. \quad (5.1)$$

On obtient la différence d'intensité de rouge entre ces deux pixels. Ainsi pour des images de couleurs, ils répète l'algorithme trois fois pour les intensités *RGB*.

La deuxième méthode pour calculer le poids entre chaque noeud revient simplement à calculer la distance dans l'espace couleur entre ces deux noeuds. Comme précédemment, on peut utiliser l'espace *LAB** pour calculer cette distance. On a alors :

$$w(p_{x+1,y}, p_{x,y}) = \sqrt{(p_{x+1,y}(L) - p_{x,y}(L))^2 + (p_{x+1,y}(A) - p_{x,y}(A))^2 + (p_{x+1,y}(B*) - p_{x,y}(B*))^2}. \quad (5.2)$$

Au final, on obtient un graphe non orienté, car chaque lien relie les pixels de manière symétrique et connexe. En effet, quel que soit le pixel, il existe un lien, ou un ensemble de liens le reliant à un autre. Le graphique 5.1 représente sous forme de graphe le célèbre tableau de Mondrian, où chaque poids entre les pixels est calculé en utilisant l'intensité couleur.

5.2 Minimum Spanning Trees

La méthode de Felzenszwalb et Huttenlocher est très proche de celle du *Minimum spanning trees* (MST) de Krugals. L'algorithme de Kruskal pour le MST ou arbre de couvrant minimal prend en donnée d'entrée un graphe non orienté et connexe. Il renvoie un arbre dit couvrant, qui connecte tous les sommets entre eux,

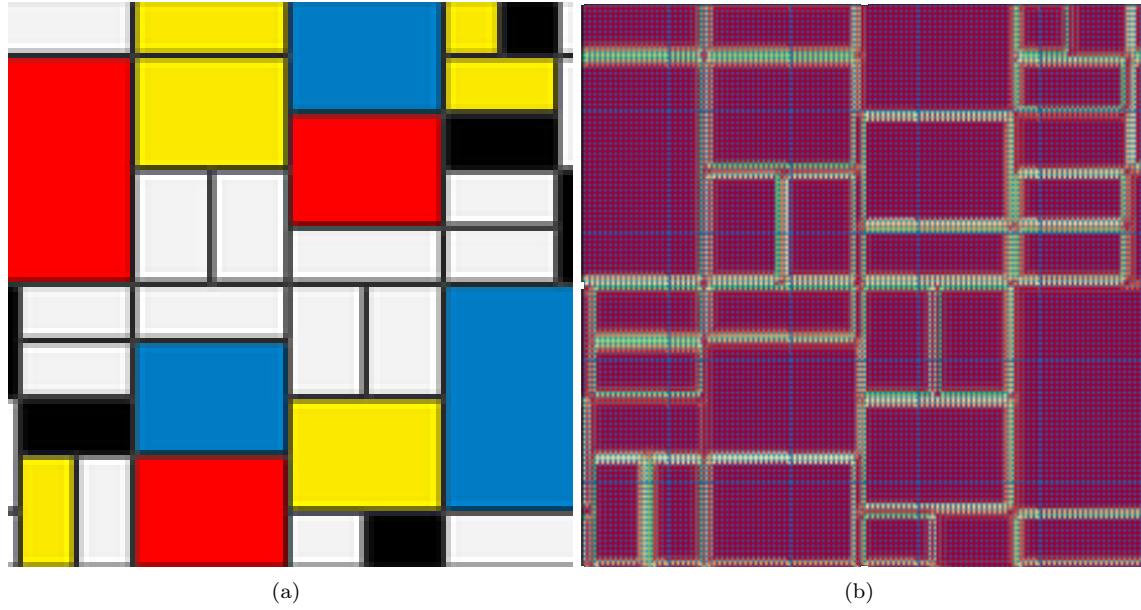


Figure 5.1: Illustration de la modélisation d'une image couleur par un graphe connexe. La couleur rouge correspond à une distance nulle et la couleur verte correspond à une distance différente de 0. Le cadrillage bleu n'est pas à prendre en compte, il correspond à un cadrillage par défaut du graph. (a) L'image d'origine est un tableau de Mondrian. (b) La représentation de cette image sous forme de graphe. On observe grâce aux changements de couleur que les poids des arêtes restent les mêmes au sein des carrés de couleurs, mais changent d'un carré à l'autre.

et dont la somme des poids est minimal. Pour bien comprendre le MST, on cite souvent « *the muddy city problem* » Murali [2009]. Dans cet exemple, on représente une ville fictitionnelle où aucun chemin reliant les maisons n'existe. Le maire décide alors de construire des routes, mais avec le moins de dépenses possible. Il s'agit donc de construire un chemin qui relie chaque maison une fois mais en construisant le moins de routes possible. L'algorithme de Kruskal permet de résoudre ce problème puisqu'il renvoie un MST reliant chaque maison entre elles, avec la somme des poids minimale : dans cet exemple, les poids correspondent à la taille des routes et les maisons correspondent aux noeuds.

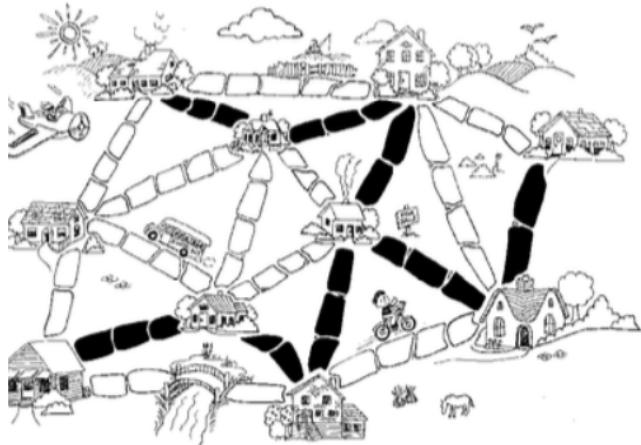


Figure 5.2: The muddy city problem

L'algorithme de Kruskal se compose alors de trois étapes pour créer un Minimum Spanning Trees (MST) :

- On commence par ranger l'ensemble des poids E du graphe dans l'ordre croissant.
- On sélectionne l'arête avec le poids le plus faible, si en ajoutant cette arête le nouveau graphe forme un cercle, alors on passe à l'arête qui a un poids supérieur, sinon on ajoute cette arête pour créer le MST.
- on arrête l'algorithme quand il y a $V-1$ arêtes dans le MST.

(Voir l'annexe A.4 pour plus d'informations sur les *Minimum spanning trees*.)

5.3 Critère pour comparer deux régions

L'algorithme de Felzenszwalb et Huttenlocher modifie la méthode de Kruskal, lors de la création du MST. Cette modification porte sur la comparaison de deux régions entre elles [Felzenszwalb and Huttenlocher, 2004]. on introduit la notation $MST(C,E)$ qui correspond au Minimum Spanning Trees du groupe de pixels C . On commence par définir "la différence interne au graphe" MST donnée par :

$$Int(C) = \max_{e \in MST(C,E)} w(e) \quad (5.3)$$

qui correspond au poids maximal du MST. L'idée intuitive est que pour une classe C on calcule le MST, $Int(c)$ est alors égal au poids maximum du MST. De plus, si le cluster n'a qu'un pixel $|C| = 1$ alors $Int(C) = 0$. On définit ensuite "la différence externe au graphe", qui correspond au poids minimal qui existe entre deux groupes de pixels C_1, C_2 parmi l'ensemble des poids reliant ces deux clusters :

$$Dif(C_1, C_2) = \min_{v_i \in C_1, v_j \in C_2, (v_i, v_j) \in E} w(v_i, v_j). \quad (5.4)$$

Si aucun lien ne relie ces deux clusters alors on fixe

$$Dif(C_1, C_2) = \infty. \quad (5.5)$$

Enfin on introduit "la différence interne minimale":

$$MInt(C_1, C_2) = \min(Int(C_1) + \frac{z}{|C_1|}, Int(C_2) + \frac{z}{|C_2|}), \quad (5.6)$$

où z est un paramètre permettant le contrôle de la taille des superpixels et $|C|$ est le nombre de pixels dans le cluster $|C|$. Plus ce paramètre est important plus $MInt(C_1, C_2)$ sera grand, et donc plus on exige une faible preuve pour modifier un cluster. Ainsi si on fixe z grand on aura une préférence pour des superpixels de grande taille. Mais attention, à l'inverse du paramètre m de l'algorithme *SLIC*, le paramètre z ne permet pas d'avoir une taille uniforme pour chaque composante (voir figure ??). Enfin on obtient la formule pour comparer deux régions :

$$D(C_1, C_2) = \begin{cases} 1 & \text{if } Dif(C_1, C_2) > MInt(C_1, C_2) \\ 0 & \text{otherwise} \end{cases} \quad (5.7)$$

Elle compare "la différence interne du graph MST" et "la différence externe" et permet de déterminer si les clusters C_1 et C_2 doivent être reliés ou non.

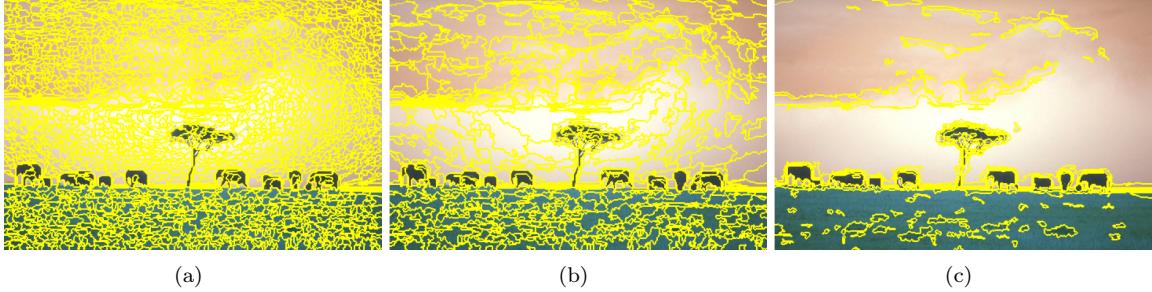


Figure 5.3: Illustration de l'impact du paramètre z sur la segmentation dans l'algorithme de FH. (a) $z = 1$
(b) $z = 10$ (c) $z = 100$

5.4 Algorithme complet.

La première étape consiste à calculer les poids entre chaque pixel comme dans la figure 5.1. Pour ce faire on calcule les poids avec la formule (5.1) ; un poids est calculé pour chaque intensité dans l'espace de couleur RGB . Ainsi au départ chaque pixel correspond à une classe individuelle C . Puis comme dans le minimum spanning trees, on ordonne les arêtes E en fonction de leur poids dans l'ordre croissant. On commence en prenant l'arête avec le plus petit poids. Puis pour chacune de ces arêtes reliant deux pixels on utilise la formule (5.8) pour comparer la différence interne à la différence externe. La formule (5.8) permet donc de relier les pixels entre eux afin de créer nos superpixels. Il est important de noter que deux régions C_1 , C_2 seront reliées entre elles uniquement si l'external difference est supérieure à l'internal difference avec les trois intensités de couleur. Ainsi, si avec les poids calculés sur l'intensité de rouge (w_r) $D(C_1, C_2) = 1$, mais que lors du calcul des poids sur l'intensité de bleu (w_b) $D(C_1, C_2) = 0$ alors il n'y aura pas de fusion entre les régions C_1 et C_2 . Quand les itérations sur toutes les arêtes prennent fin, l'algorithme s'arrête et nous renvoie les superpixels créés.

Chapter 6

Comparaison de méthodes

Concernant la base de données, les images utilisées ont exclusivement été prises dans le Dataset de Berkley, Martin et al. [2001]. J'y ai sélectionné 108 images couleurs et leur vérité terrain associée. A l'issue de cette dernière partie je mesure les performances des algorithmes de segmentation en utilisant les métriques présentées dans la section 2.3.

6.1 L'importance du paramétrage dans SLIC.

D'abord, on observe l'influence des paramètres M et K sur la compacité equation (2.7), l'homogénéité des couleurs d'un superpixel equation (2.1) et l'adhérence aux frontières de l'image equation (2.3). Pour ce faire, je mesure ces métriques sur les 108 images en faisant varier les paramètres. Pour rappel, la compacité se mesure par la métrique SRC : plus elle est proche de 1 plus les formes des superpixels sont similaires. L'homogénéité des couleurs se mesure par la métrique EV : une valeur importante de l'EV indique que les couleurs sont fortement homogènes au sein du superpixel. L'adhérence à la frontière de l'image est quant à elle déterminée par l'UE : plus sa valeur est faible plus les superpixels respectent les frontières de l'image. Les trois métriques prenant leurs valeurs entre 0 et 1, il est possible de les représenter dans un même graphique afin d'observer les tendances.

Pour tester l'influence de la compacité, je fixe le nombre de superpixels à 200 et je fais varier le paramètre m de 1 à 120. Les résultats sont présentés dans la figure ci-dessous.

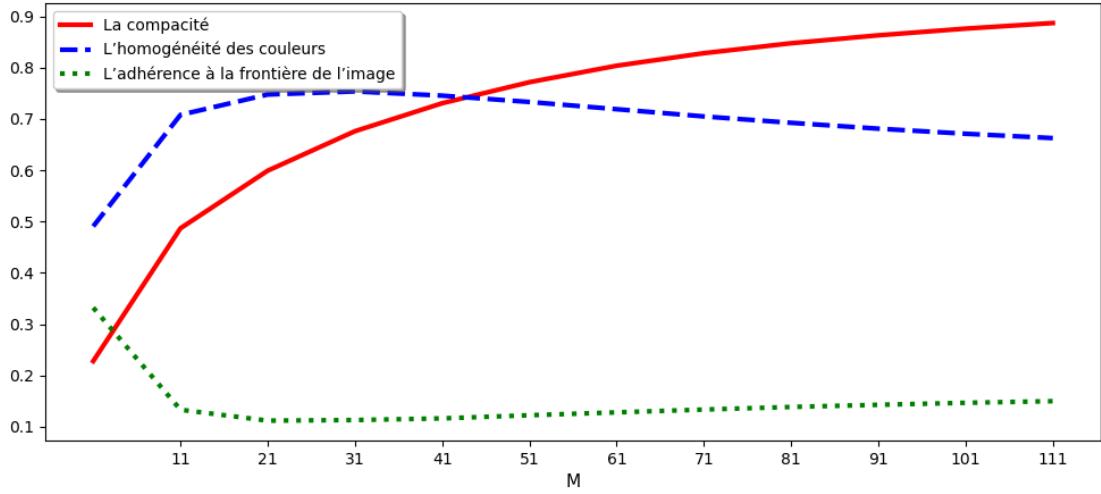


Figure 6.1: Algorithme SLIC calculé sur 108 images, pour mesurer la compacité (SRC), l'homogénéité des couleurs (EV) et l'adhérence à la frontière de l'image (UE2) avec $K = 200$ et $M \in \{1, 2, \dots, 120\}$.

Comme on pouvait s'y attendre, la compacité augmente avec le paramètre m . On voit également qu'en augmentant m au-delà de 20, on commence à perdre en homogénéité des couleurs et en adhérence à la frontière de l'image. Pour cette raison, paramétriser la valeur de m entre 10 et 20 est privilégié.

Afin de tester l'influence du nombre de superpixels sur les propriétés dans cette deuxième expérience, je fixe le paramètre m à 10 et fais varier le nombre de superpixels générés par le SLIC de 10 à 400.

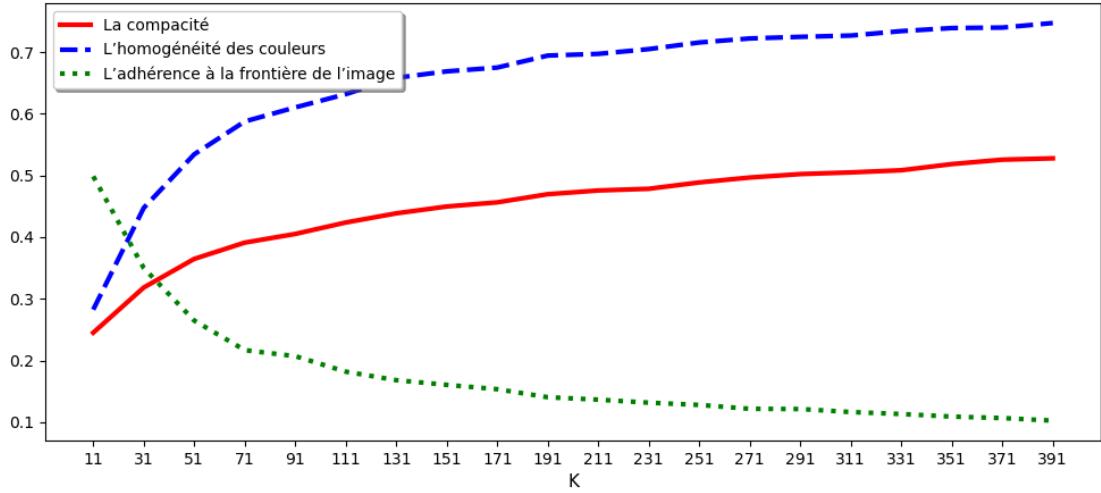


Figure 6.2: Algorithme SLIC calculé sur 108 images, pour mesurer la compacité (SRC), l'homogénéité des couleurs (EV) et l'adhérence à la frontière de l'image (UE2) avec $M = 10$ et $K \in \{10, 11, \dots, 400\}$.

Dans le graphique 6.2, on voit clairement qu'augmenter le nombre de superpixels permet d'augmenter toutes les métriques. Seulement cette augmentation sera de moins en moins importante à mesure qu'on augmente le nombre de superpixels. C'est l'une des grandes forces de l'algorithme SLIC, avec une valeur importante du K , on s'assure de ne quasiment pas perdre d'information.

6.2 L'importance du paramétrage dans FH.

Par la suite, je mesure l'influence du paramètre z sur les propriétés de cet algorithme. Comme pour la section précédente, j'utilise la base des 108 images et calcule pour chaque valeur de z les trois métriques. Les résultats sont disponibles dans la figure 6.3.

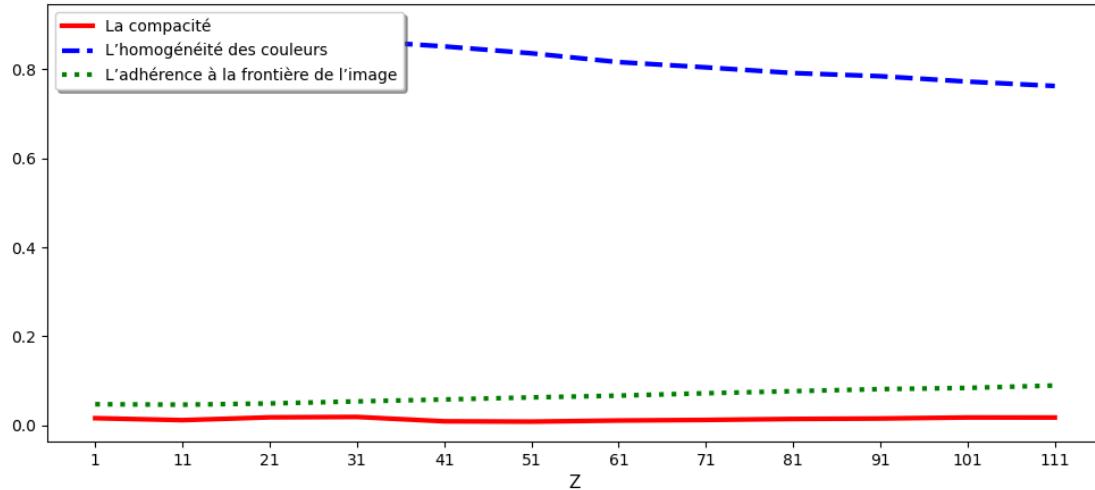


Figure 6.3: Algorithme FH calculé sur 108 images, pour mesurer la compacité (SRC), l'homogénéité des couleurs (EV) et l'adhérence à la frontière de l'image (UE2) avec $M = 10$ et $K \in \{10, 11, \dots, 400\}$.

La première chose que l'on observe est la faible influence du paramètre z sur les trois propriétés. La compacité mesurée avec le *SRC* est très mauvaise, qu'elle que soit la valeur de z . En effet, comme le montre la figure 5.3, les superpixels générés par cet algorithme sont très éloignés du carré et sont même des lignes dans certains cas. Ce score est faible en raison de leur forme. Néanmoins si on observe les deux autres courbes, ces dernières montrent que cet algorithme conserve une grande quantité d'informations puisqu'il obtient un très bon score sur l'homogénéité des couleurs (quasiment toujours 0.8) mais également un très bon score sur l'adhérence aux frontières de l'image (0.1). Ainsi, l'algorithme FH permet de conserver plus d'informations que l'algorithme SLIC car son score est quasiment toujours supérieur sur les mesures d'homogénéité des couleurs et d'adhérence à la frontière de l'image.

6.3 Comparaison de la segmentation par graphe et de SLIC.

Nous comparons maintenant les performances de ces deux algorithmes et examinons sous quelles conditions l'un sera préféré à l'autre. La première différence provient du fait que l'algorithme FH ne permet pas la sélection du nombre de superpixels générés. Il rend alors impossible une comparaison des deux algorithmes sur ce critère.

Comme expliqué en section 2.3, il est également important pour un algorithme de segmentation en superpixels d'être rapide. J'effectue donc toujours sur les 108 images une moyenne du temps de segmentation en fonction des paramètres k, z, m , les résultats sont visibles sur la figure 6.4. On voit alors que le temps de la segmentation est indépendant de ces paramètres. Cela confirme que contrairement au K -means, l'algorithme SLIC est de complexité $O(N)$. On observe également que l'algorithme SLIC est légèrement plus rapide (0.1 seconde de moins) que l'algorithme FH, quel que soit le paramétrage utilisé. Néanmoins ce gain est si faible que le choix entre l'algorithme de Felzenwalb ou le SLIC ne reposera sûrement pas sur ce critère.

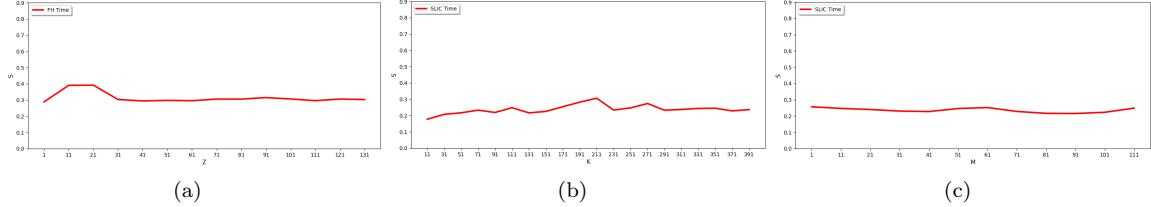


Figure 6.4: Illustration de l'impact du paramètre z sur la segmentation dans l'algorithme de FH. (a) $z = 1$
(b) $z = 2$ (c) $z = 3$

Nous avons vu deux algorithmes assez différents, disposant chacun de leurs avantages et de leurs inconvénients. Le choix entre ces deux algorithmes dépend du format de superpixel voulu. Si on veut pouvoir choisir le nombre de superpixels et leur compacité, on préférera l'algorithme SLIC qui laisse plus de liberté. Si on cherche seulement à générer des superpixels sans perdre d'information sur l'image, on choisira alors l'algorithme de Felzenszwalb et Huttenlocher. On peut étendre ce raisonnement à l'ensemble des algorithmes de segmentation en superpixel, il n'y a sûrement pas d'algorithmes ayant le score maximal sur l'ensemble des propriétés. Ainsi le choix d'un algorithme de segmentation en superpixel doit avant tout dépendre de l'application que l'on souhaite en faire.

Chapter 7

Conclusion et perspectives

L'objectif de ce mémoire était d'avoir une vision globale de la segmentation en superpixels. Cela a été réalisé en décrivant précisément le fonctionnement de deux algorithmes. Le SLIC d'une part qui a un fonctionnement très similaire au K -means mais vient en modifier certains aspects afin de s'adapter à la segmentation en superpixels. L'algorithme de Felzenszwalb et Huttenlocher d'autre part ayant la particularité d'être basé sur la théorie du partitionnement des données par graphe. Pour compléter cette analyse théorique de la segmentation en superpixel, une analyse empirique a été menée pour observer le comportement de ces algorithmes sur de vraies images. Pour ce faire une définition des propriétés que doit respecter un superpixel, ainsi que des métriques pour évaluer ces dernières ont été fournies. Tous ces éléments ont finalement permis d'effectuer une étude comparative.

Grâce à cette méthodologie il a été possible de dégager les avantages et inconvénients des deux algorithmes. Pour le SLIC son avantage vient de son paramétrage. En prenant un K (200 ou plus) suffisamment grand, et un m compris entre 10 et 20, on s'assure de conserver une grande partie de l'information contenue dans l'image tout en générant des superpixels compacts. Pour FH, cette méthodologie a d'abord mis en évidence sa capacité à conserver une quantité importante d'information et ce quel que soit le paramètre z . Néanmoins, FH souffre du fait qu'on ne puisse choisir le nombre de superpixels, et que ceux-ci sont loin d'être compacts. Finalement on observe que le choix de SLIC ou de FH dépend avant tout des propriétés que l'on souhaite optimiser.

En ce sens, pour compléter ce mémoire il aurait été intéressant d'observer comment ces deux algorithmes se comportent en étape de pré-traitement sur des applications de vision par ordinateur. Cette étape aurait permis de comprendre quelles sont les propriétés les plus fondamentales à respecter en fonction de la tâche à accomplir. Mais également de prouver que l'utilisation des superpixels à la place des pixels peut permettre d'augmenter les performances de certaines applications Achanta et al. [2010].

Appendix A

Annexes

A.1 Les espace de couleur RGB, LAB

Il existe un grand nombre d'espaces de couleurs différentes. C'est la Commission Internationale de l'Éclairage (CIE) qui est en charge de définir les espaces de couleurs comme le *RGB* ou le *LAB**. Ces deux espaces ont été utilisés durant ce stage et ont une définissent la couleur différemment. Le *RGB* est un codage de la couleur se basant sur les trois intensités lumineuses Rouge, Vert (green) et Bleu, chaque intensité pouvant aller d'une valeur de 0 à 255. L'espace de couleurs *LAB** est lui dit à « luminance séparée »Boyer [2013]. En effet le *L* correspond à la luminance en pourcentage avec 0 pour le noir et 100 pour le blanc. Alors que le *A* et *B** sont deux gammes de couleurs, le *A* allant du vert au rouge et le *B** du Bleu au jaune. Le *A* et *B** pouvant prendre des valeurs allant de -120 à +120. Il est possible de passer du *RGB* au *LAB** comme ceci Acharya and Ray [2005] :

$$\begin{aligned} X &= 0.412453R + 0.357580G + 0.180423B \\ Y &= 0.212G71R + 0.715160G + 0.0721G9B \\ Z &= 0.019334R + 0.119193G + 0.950227B \end{aligned}$$

$$\begin{aligned} L &= 116f(Y/Y_n) - 16 \\ A &= 500[f(X/X_n) - f(Y/Y_n)] \\ B* &= 200[f(Y/Y_n) - f(Z/Z_n)] \end{aligned}$$

$$f(q) = \begin{cases} q^{1/3} & \text{si } q > 0.08856 \\ 7.787q + 16/116 & \text{sinon} \end{cases}$$

A.2 Exemple de vérité terrain

Durant ce stage j'ai principalement utilisé des images du Dataset de Berkley Martin et al. [2001]. Dans ce dataset on trouve notamment des vérités terrain, c'est à dire une segmentation déjà réalisée. Ces vérités terrain sont utilisées dans la métriques sur l'adhérence à la frontière de l'image. J'ai mis quatre images du dataset et leur vérité terrain sur la figure A.1.

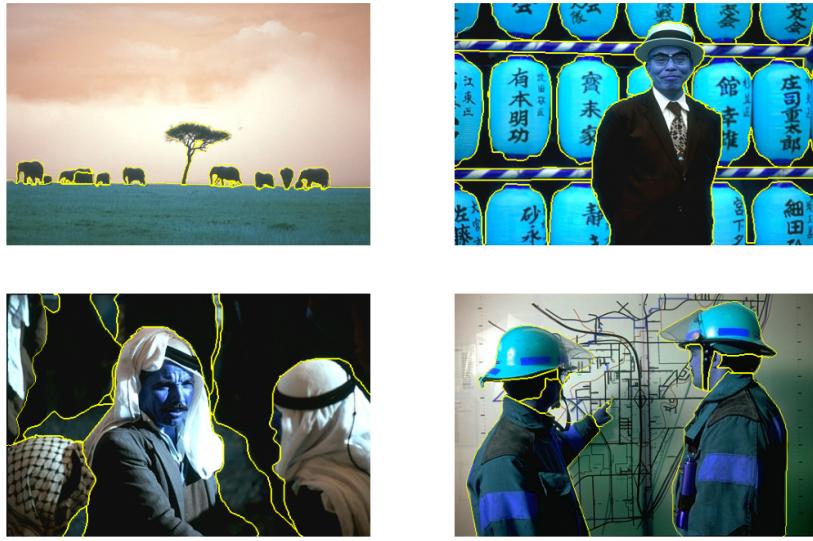


Figure A.1: Exemple de quatre images et leur vérité terrain en jaune.

A.3 Exemple d'enveloppe convexe

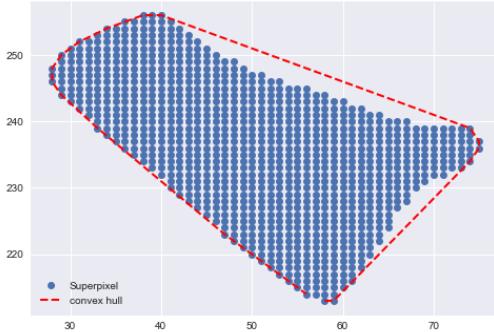


Figure A.2: Le nuage de points correspondant aux pixels du superpixel et les pointillés rouge à l'enveloppe convexe.

A.4 Exemple de Minimum Spanning Trees

Dans l'exemple ci-dessous on part d'un graphe composé de 9 noeuds et on cherche son Minimum Spanning Tree. Le MST qui en découle a 8 liens et tous les noeuds sont reliés, aucun cercle ne s'est formé et la somme des liens est minimale.

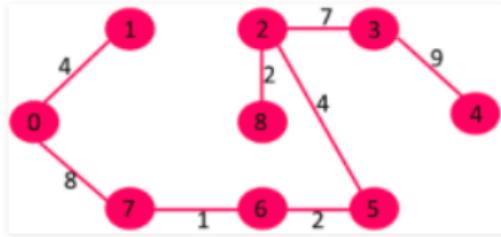
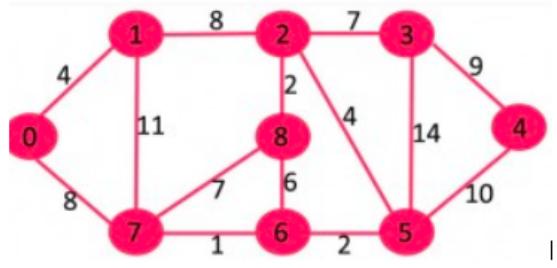


Figure A.3: Exemple de MST

Bibliography

- Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Süsstrunk. Slic superpixels. Technical report, 2010.
- Tinku Acharya and Ajoy K Ray. *Image processing: principles and applications*. John Wiley & Sons, 2005.
- Zhihua Ban, Jianguo Liu, and Li Cao. Superpixel segmentation using gaussian mixture model. *IEEE Transactions on Image Processing*, 27(8):4105–4117, 2018.
- Christopher M Bishop. *Pattern recognition and machine learning*. Springer, 2006.
- Elise Arnaud Edmond Boyer. Analyse d’images. INRIA Rhône-Alpes, 2013. http://morpheo.inrialpes.fr/people/Boyer/Teaching/L3/L3_analyse.pdf.
- Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient graph-based image segmentation. *International journal of computer vision*, 59(2):167–181, 2004.
- Daniela Giordano, Francesca Murabito, Simone Palazzo, and Concetto Spampinato. Superpixel-based video object segmentation using perceptual organization and location prior. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4814–4822, 2015.
- Rémi Giraud, Vinh-Thong Ta, and Nicolas Papadakis. Evaluation framework of superpixel methods with a global regularity measure. *Journal of Electronic Imaging*, 26(6):061603, 2017.
- Zhongwen Hu, Qin Zou, and Qingquan Li. Watershed superpixel. In *2015 IEEE International Conference on Image Processing (ICIP)*, pages 349–353. IEEE, 2015.
- Alex Levinstein, Adrian Stere, Kiriakos N Kutulakos, David J Fleet, Sven J Dickinson, and Kaleem Siddiqi. Turbopixels: Fast superpixels using geometric flows. *IEEE transactions on pattern analysis and machine intelligence*, 31(12):2290–2297, 2009.
- D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proc. 8th Int'l Conf. Computer Vision*, volume 2, pages 416–423, July 2001.
- TM Murali. Applications of minimum spanning trees, 2009.
- Peer Neubert and Peter Protzel. Superpixel benchmark and comparison. In *Proc. Forum Bildverarbeitung*, volume 6, pages 1–12, 2012.
- David Pollard. Strong consistency of k-means clustering. *The Annals of Statistics*, pages 135–140, 1981.
- Alexander Schick, Mika Fischer, and Rainer Stiefelhagen. Measuring and evaluating the compactness of superpixels. In *Proceedings of the 21st international conference on pattern recognition (ICPR2012)*, pages 930–934. IEEE, 2012.
- Murong Wang, Xiabi Liu, Yixuan Gao, Xiao Ma, and Nouman Q Soomro. Superpixel segmentation: A benchmark. *Signal Processing: Image Communication*, 56:28–39, 2017.
- Fan Yang, Huchuan Lu, and Ming-Hsuan Yang. Robust superpixel tracking. *IEEE Transactions on Image Processing*, 23(4):1639–1651, 2014.
- Ian T Young, Jan J Gerbrands, and Lucas J Van Vliet. Fundamentals of image processing. 1998.