Université de Lille
Faculté des Sciences et Technologies

Algorithmes de segmentation d'images non supervisés appliqués à la détection d'objets

Stevenson PATHER

Master Machine Learning
Projet Individuel

Département d'Informatique
Faculté des Sciences et Technologies

juin, 2023

Etudiant: Stevenson PATHER, stevenson-pather.etu@univ-lille.fr

Encadrante: Deise Santana Maia, deise.santanamaia@univ-lille.fr

Département d'Informatique
Faculté des Sciences et Technologies
Campus Cité Scientifique, Bât. M3 extension, 59655 Villeneuve-d'Ascq

juin, 2023

Remerciements

Je tiens à remercier personnellement Mme Deise Santana Maia pour la qualité de son encadrement durant ce projet de master. Les réunions et différents échanges que j'ai pu avoir avec elle m'ont permis de mener à bien ce projet dans de bonnes conditions. En plus d'une expérience enrichissante, j'ai pu découvrir le domaine de la segmentation d'image et améliorer mes compétences nécessaire pour mener à bien un projet de recherche.

Indice

Table des figures

Chapitre 1

Introduction

La segmentation d'image est utilisée pour la détection d'objets ou de contours, et permet d'avoir une meilleure compréhension plus fine des images et des objets qui la constituent. Encore aujourd'hui, les problèmes de segmentation et de regroupement d'images restent de grands défis dans le domaine de la vision par ordinateur, mais également dans l'imagerie médicale, l'analyse d'images satellitaires ou encore les voitures autonomes. Ainsi l'apport de méthodes de calcul de segmentations fiables et efficaces peut bénéficier à de nombreuses problématiques dans divers domaines d'application.

La segmentation d'images est une technique qui permet de détecter et regrouper les pixels d'une image suivant certains critères, notamment d'intensité ou spatiaux, l'image après segmentation se révèle ainsi formée de régions uniformes. Les méthodes de segmentation d'images ont longuement reposées sur des techniques de détection de contours classiques, par l'utilisation d'opérateurs mathématiques simples comme le gradient ou la dérivée pour détecter des pixels ou des changements brusque d'intensité. Nous retrouvons ainsi les approches basées région (découpe, fusion, découpe/fusion), les approches basées contour ou d'autres approches comme Mumford Shah, modèles déformables, level sets ou champs de Markov. Nous nous intéressons dans ce document à deux méthodes, la méthode de Felzenszwalb et Huttenlocher ainsi que la méthode par watershed, pour comparer et évaluer la qualité des différentes segmentations obtenues sur plusieurs catégories d'images.

Chapitre 2

Méthode de Felzenszwalb et Huttenlocher

Le but de rechercher à l'époque par Pedro F. Felzenszwalb et Daniel P. Huttenlocher dans leur publication Efficient Graph-Based Image Segmentation [1] est de proposé une approche avec les propriétés suivantes :

— La segmentation doit capturer des composantes ou des régions visuellements importantes qui représentent des aspects globaux de l'image,

— La méthode doit être efficace, donc avoir une complexité en temps linéaire par rapport aux dimensions de l'image : $\mathcal{O}(width * height)$.

Bien que des méthodes aient fait des progès dans la segmentation d'images, dans l'état de la littérature au moment de la publication de l'article, la plupart d'entre elles étaient lentes et ne tenaient pas compte des objets avec une invariance d'intensité.

2.1 Une approche basée sur les graphes

Pour pallier ceci Felzenszwalb et Huttenlocher ont construit leur méthode en utilisant une approche de segmentation basée sur les graphes. Ainsi le problème de segmentation d'une image peut être décrit de la façon suivante, soit $G = (V, E)$ un graphe indirect avec pour sommets $v_i \in V$ l'ensemble des pixels à segmenter, et les

arêtes $(v_i, v_j) \in E$ l'ensemble correspondant à des paires de sommets voisins. Ainsi pour chaque arête $(v_i, v_j) \in E$ le poids associé $w((v_i, v_j))$ correspond à la mesure de dissemblance entre chaque pixels voisins (v_i, v_j). Cette mesure de dissemblance peut prendre plusieurs formes, différence d'intensité, de couleur, de position ou de tout autre attribut local. Nous reviendrons sur la mesure utilisée dans cette méthode pour la définir de manière formelle. Nous pouvons ainsi définir une segmentation S d'un tel graphe G où $G' = (V, E')$ où $E' \subset E.S$ divise G en G' tel que les composantes C' de G' soient strictement distincts des composantes C de G[1]. Pour construire un tel graphe G à partir d'une image donnée deux approches sont possibles.

2.1.1 Grid graph

Dans cette approche on considère l'image et le graphe comme une grille, ainsi chaque pixel est connecté avec ses voisins directs, donc huits au total. Et chaque arête a un poids correspondant à la valeur absolue de la différence d'intensité entre chaque pairs de pixels voisins.

2.1.2 Nearest Neighbor Graph

L'approche par plus proches voisins consiste à representer chaque pixel comme un point de l'espace par (x, y, r, g, b), où (x, y) correspond à la position du pixel dans l'espace de l'image, et (r, g, b) les valeurs de couleur RVB. Et le poids des arêtes est la distance euclidienne entre chaque pairs de pixels du graphe.

2.2 Segmentation à partir du graphe

Une fois le graphe formé à partir d'une image, nous pouvons l'utiliser pour obtenir une segmentation de l'image à l'aide d'un prédicat de segmentation qui consistuera le coeur de l'algorithme de segmentation donné par Felzenszwalb et Huttenlocher.

2.2.1 Une composante

Dans un premier temps, on définit ici une composante comme :

Définition 1 *Une composante est une composante connexe du graphe, soit un sous-graphe connexe de ce graphe, composée par un ensemble de pixels celle-ci représente une région segmentée de l'image.*

2.2.2 Différence entre deux composantes

On définit la différence entre deux composantes par l'arête de poids minimum qui connecte les deux composantes :

$$Dif(C_1, C_2) = \min_{v_1 \in C_1, v_j \in C_2, (v_i, v_j) \in E} w((v_i, v_j)) \tag{2.1}$$

2.2.3 Différence interne d'une composante

La différence interne dans une composante est donnée par l'arête de poids maximum qui connecte deux sommets d'une même composante :

$$Int(C) = \max_{e \in MST(C,E)} w(e) \tag{2.2}$$

où *Minimum Spanning Tree* (MST) est l'arbre couvrant de poids minimal des composantes. Ainsi une composante C peut toujours rester connectée même lorsque toutes les arêtes $(v_i, v_j) \in E < Int(C)$ ont été retirées.

2.2.4 Fonction de seuil τ

Pour les petites composantes le calcul de $Int(C)$ n'est pas une bonne estimation des caractéristiques locales des données. Dans des cas extrêmes on obtient $|C| = 1 \implies Int(C) = 0$. Ainsi, la méthode utilise une fonction de seuil basée sur la cardinalité des composantes :

$$\tau(C) = \frac{k}{|C|} \tag{2.3}$$

où $|C|$ est la cardinalité de la composante C, et k une constante donnée.

La fonction de seuil τ va permettre le contrôle du degré auquel la différence entre deux composantes doit être supérieurs à leurs différences internes pour que l'on considère qu'il y ait une frontière entre les composantes[1].Ce contrôle est nécessaire dans le calcul de la différence interne minimale d'une composante. Ainsi au plus k sera faible au plus il sera difficile de pouvoir fusionner des composantes entre elles donc nous aurons plus de régions dans la segmentation avec un k faible.

2.2.5 Différence interne minimale entre deux composantes

La différence interne minimale entre deux composantes C_2 et C_2 est défini par :

$$MInt(C_1, C_2) = \min\left(Int(C_1) + \tau(C_1), Int(C_2) + \tau(C_2)\right) \tag{2.4}$$

C'est donc la différence interne des composantes C_1 et C_2 contrôlé par la constante k. Comme exprimé précédemment $\tau(C)$ définit le seul par lequel les composantes doivent être différents des noeuds internes donc des pixels d'une autre composante[1] pour permettre ou non l'aggrégation de celles-ci en une seule composante. Ainsi les

propriétés de la constante k sont les suivantes :

— un k élevé aura tendance à former des régions plus grandes,

— k n'apporte aucune contrainte de taille minimale pour les régions.

2.2.6 Prédicat de segmentation

Finalement pour mesurer la qualité de la segmentation, la méthode utilise un prédicat de comparaison de régions par paires pour deux régions données C_1 et C_2 :

$$D(C_1, C_2) = \begin{cases} true & \text{if } Dif(C_1, C_2) > MInt(C_1, C_2) \\ false & \text{otherwise} \end{cases} \tag{2.5}$$

Ainsi deux composantes sont considérées comme indépendantes si et seulement si le prédicat ci-dessus est vrai. Auquel cas la segmentation est certainement trop fine et les composantes doivent être fusionnées. Donc ce n'est que lorsque la différence entre deux composantes est supérieure à la différence interne minimale des deux composantes que celles-ci peuvent être divisées en deux composantes distinctes. Nous pouvons résumer l'utiliser de $MInt$ et Dif par le prédicat avec cet exemple ci-dessous :

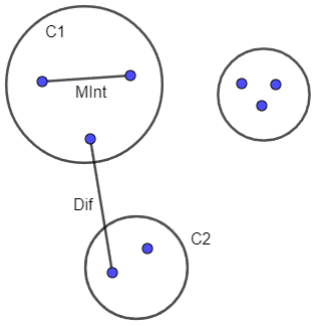


FIGURE 2.1 – Prédicat de segmentation

2.3 L'algorithme et ses propriétés

Maintenant que le prédicat de segmentation est définit, nous pouvons décrire et analyser l'algorithme de Felzenszwalb qui en utilisant le critère de décision D ci-dessus permet de produire une segmentation.

2.3.1 L'algorithme de Felzenszwalb

L'entrée est un graphe $G = (V, E)$, avec n sommets et m arêtes. Le résultat est une segmentation de V en composantes $S = (C_1, ..., C_r)$.

Algorithme 1 *algorithme de Felzenszwalb*

0. Trier E en $\pi = (o_1, ..., o_m)$, par poids d'arête non décroissant.

1. Commencer avec une segmentation S_0, où chaque sommet v_i est dans sa propre composante.

2. Répétez l'étape 3 pour $q = 1, ..., m$.

3. Construire S^q étant donné S^{q-1} comme suit. Soient v_i et v_j les sommets connectés par la q-ième arête dans l'ordre, c'est-à-dire $o_q = (v_i, v_j)$. Si v_i et v_j sont dans des composantes disjointes de S^{q-1} et que $w(o_q)$ est plus petit que la différence interne de ces deux composantes, alors fusionner les deux composantes sinon ne rien faire. Plus formellement, soit C_i^{q-1} la composante de S^{q-1} contenant v_i et C_j^{q-1} la composante contenant v_j. Si $C_i^{q-1} \neq C_j^{q-1}$ et que $w(o_q) \leq MInt(C_i^{q-1}, C_j^{q-1})$ alors S^q est obtenu à partir de S^{q-1} en fusionnant C_i^{q-1} et C_j^{q-1}. Sinon $S^q = S^{q-1}$.

4. Retourner $S = S^m$

On peut constater que l'algorithme suit une porocédure dite bottom-up et qu'il tourne autour du prédicat de segmentation. On peut reprendre l'algorithme pour décrire les étapes suivantes étant donner un graphe $G = (V, E)$, $|V| = n$ et $|E| = m$ où $|V|$ est le nombre de sommets (pixels) et $|E|$ est le nombre d'arêtes :

1. les arêtes sont triées par poids dans l'ordre croissant, étiquetées comme $e_1, e_2, ..., e_m$.

2. Initialement, chaque pixel reste dans sa propre composante il y a donc n composantes au début.

3. Répétez pour $k = 1, ..., m$:

- La segmentation à cette étape k est noté S^k
- Nous prenons la k-ième arête dans l'ordre, $e_k = (v_i, v_j)$.
- Si v_i et v_j appartiennent à la même composante ($S^k = S^{k-1}$), alors ne rien faire
- Si v_i et v_j appartiennent à deux composantes différentes C_i^{k-1} et C_j^{k-1} comme dans la segmentation S^{k-1} nous les fusionnons en une seule composante si $w(v_i, v_j) \leq MInt(C_i^{k-1}, C_j^{k-1})$, sinon ne rien faire

Voici l'algorithme appliqué à un exemple visuel :

FIGURE 2.2 – Tous les pixels sont une composante et recherche de l'arête de poids minimum entre deux composantes

FIGURE 2.3 – Fusion des deux composantes et recherche de l'arête de poids minimum suivante entre deux composantes

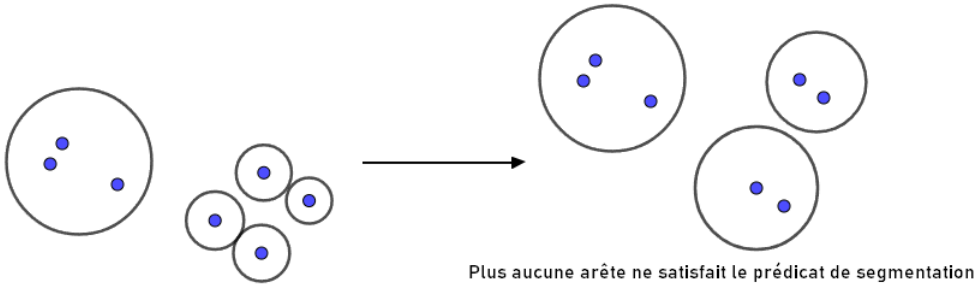


FIGURE 2.4 – Fusion des deux composantes suivantes et prédicat non satisfait

Ainsi l'algorithme s'arrête dès lors que le prédicat de segmentation D n'est plus satisfait pour aucune paires de composantes différentes.

2.3.2 Les Propriétés

Felzenszwalb et Huttenlocher précisent qu'une segmentation produit par leur algorithme est ni trop grossière ni trop fine selon les définitions 1 et 2 qu'ils donnent sur les segmentations[1]. Ce qui permet de définir la propriété suivante :

Propriété 1 *Pour tout graphe (fini) $G = (V, E)$ il existe une segmentation S qui n'est ni trop grossière ni trop fine*

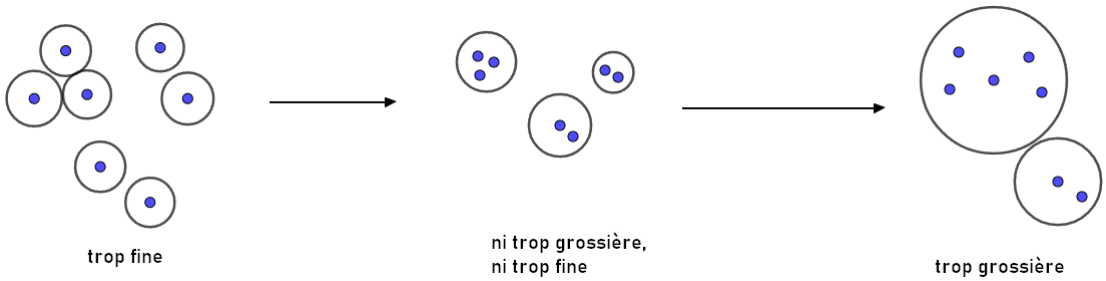


FIGURE 2.5 – Segmentation ni trop grossière ni trop fine

Chapitre 3

Méthode par watershed (LPE)

Cette technique consiste à faire grossir simultanément toutes les régions jusqu'à ce que l'image soit entièrement segmentée. Elle tire son nom d'une analogie avec la géophysique *Ligne de partage des eaux* (LPE), watershed en anglais. On peut en effet considérer les valeurs d'intensité des pixels d'une image comme une information d'altitude. Dans ce cas on peut représenter cette image (appelée carte d'élévation) comme un terrain en 3 dimensions. Le principe est alors de remplir progressivement d'eau chaque bassin du terrain. Chaque bassin représente une région. Lorsque l'eau monte et que deux bassins se rejoignent, la ligne de rencontre (i.e. la ligne de partage des eaux) est marquée comme une ligne de frontière entre les deux régions.

La méthode par watershed que l'on utilise est l'approche proposée dans le document[2] écrit par Jean Cousty, Laurent Najman, Yukiko Kenmochi et Silvio Guimarães qui utilisent des hiérachie de zones quasi-plates, des MST et des cartes de saillance pour obtenir une segmentation par watershed.

3.1 Hiérarchie connectée de partitions

Dans les sections suivantes, nous donnerons que les définitions de base pour gérer les partitions, les hiérarchies et la connectivité basées sur des graphes. Pour approfondir ces notions de référer à la publication de J. Cousty et al.[2].

Définition 2 *Une partition d'un ensemble fini V est un ensemble P de sous-ensembles non vides de V dont l'union est V, c'est-à-dire que $\forall X, Y \in P, X \cap Y = \emptyset$ si $X \neq Y$ et que $\cup\{X \in P\} = V$. Tout élément d'une partition P de V est appelé une région de P.*

Définition 3 *Une hiérarchie sur V est une séquence $\mathcal{H} = (P_0, ..., P_\ell)$ de partitions de V telle que $[P]_i \forall i \in \{1, ..., \ell\}$.*

Si $\mathcal{H} = (P_0, ..., P_\ell)$ est une hiérarchie, l'entier ℓ est appelé la profondeur de $\mathcal{H}$. Une hiérarchie $\mathcal{H} = (P_0, ..., P_\ell)$ est dite complète si $P_\ell = \{V\}$ et si P_0 contient tous les singletons de V, autrement dit $P_0 = \{\{x\} | x \in V\}$. Les hiérarchies dans notre document seront considérées comme complètes.

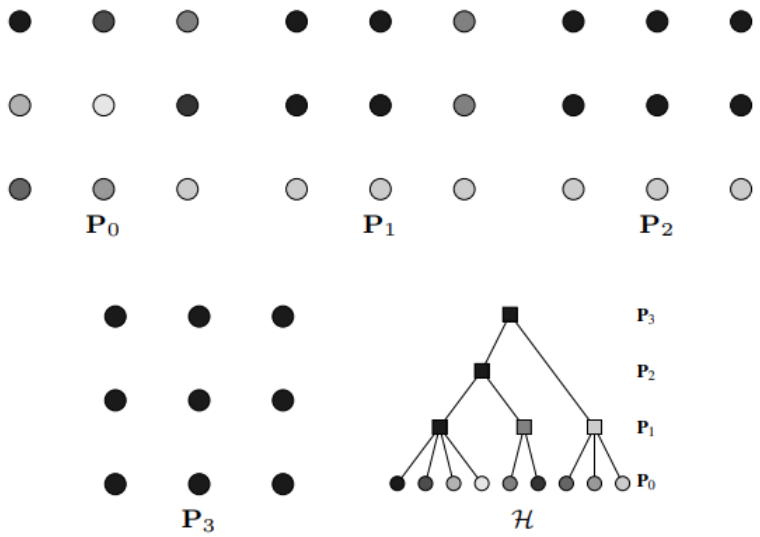


FIGURE 3.1 – Illustration d'une hiérarchie et de ses partitions

Pour chaque chaque partition, chaque région est représentée par un niveau de gris : deux deux points ayant le même niveau de gris appartiennent à la même région. La dernière dernière sous-figure représente la hiérarchie H sous la forme d'un arbre, souvent appelé dendrogramme, où la relation d'inclusion entre les régions des partitions successives est représentée par des lignes. des partitions successives est représentée par des segments de ligne[2].

Définition 4 *Etant donnée un graphe $G = (V, E)$, une partition de V est connexe (pour G) si chacune de ses régions sont connexes, et une hiérarchie sur V est connexe (pour G) si chacune de ses partitions sont connexes.*

3.2 Hiérarchie de zones quasi-plates

La publication de J. Cousty[2] nous montre qu'une hiérarchie connexe peut être traitée de manière équivalente au moyen d'un graphe pondéré (par les arêtes), sachant que pour tout graphe pondéré leur ensembles de niveau induit une hiérachie de zones quasi-plates, On nomme parfois cette structure de données α-tree.

Soit G un graphe, si w est une carte de l'ensemble des arêtes de G vers l'ensemble R^+ des nombres réels positifs, alors la paire (G, w) est appelée un graphe pondéré. Si (G, w) est un graphe pondéré par les arêtes et pour toute arête u de G, la valeur $w(u)$ est le poids de u (pour w).[2]

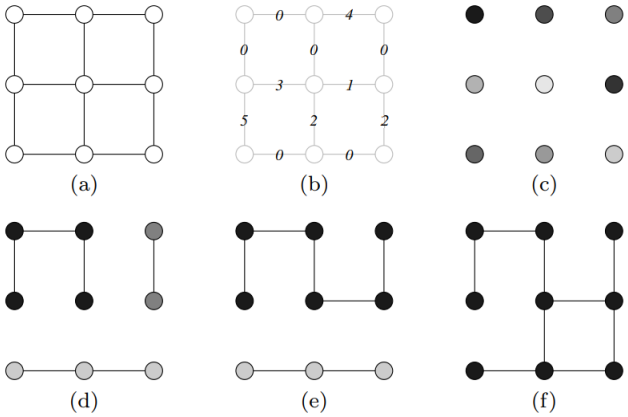


FIGURE 3.2 – Illustration d'une hiérarchie de zones quasi-plates

Ainsi on peut observer sur cette illustration d'une hiérarchie de zones quasi plates :

— un graphe G (a),
— une carte w (chiffres en noir) qui pondère les arêtes de G (en gris) (b),
— le graphe à λ niveaux de G, avec $\lambda = 0, 1, 2, 3$ (c, d, e, f).

Les partitions de composantes connectées associées qui constituent la hiérarchie des zones quasi-plates de G pour w sont représentées sur la Figure 3.1.

Définition 5 *Hiérarchie de zones quasi-plates*

Soit X un sous-graphe de G, si λ_1 et λ_2 sont deux éléments dans $\mathbb{E} = \{0, ..., |E| - 1\} \cup \{|E|\}$ tel que $\lambda_1 \leq \lambda_2$, on peut voir que pour toute arête du graphe au niveau λ_1 de X est aussi une arête du graphe au niveau λ_2 de X. Ainsi, si deux points sont connectés pour le graphe au niveau λ_1 de X, alors ils le sont aussi pour le graphe au niveau λ_2 de X. De fait, toute composante connectée du graphe au niveau λ_1 de X est également connectée pour le graphe au niveau λ_2 de X. Donc la partition de X au niveau λ_1 est un raffinement de la partition de X au niveau λ_2. Par conséquent, la séquence de toutes les partitions de X de niveau λ est une hiérarchie. Cette hiérarchie $\mathbb{QFZ}(X, w)$ est appelée hiérarchie des zones quasi-plates de X (pour w)[2] :

$$\mathbb{QFZ}(X, w) = (C(w_\lambda^V(X))|\lambda \in \mathbb{E}) \tag{3.1}$$

Nous partons d'une image en niveaux de gris, en prenant le gradient de celle-ci. Ainsi le poids de $w(u)$ sera la différence absolue d'intensité entre x et y.

3.3 Carte de saillance

La carte de saillance est une image dans laquelle la luminosité d'un pixel représente le degré de saillance du pixel d'une image donnée, c'est-à-dire que la luminosité d'un pixel est directement proportionnelle à sa saillance. Il s'agit généralement d'une image en niveaux de gris. Les cartes de saillance sont également appelées cartes de chaleur, la chaleur faisant référence aux régions de l'image qui ont un impact important sur la prédiction de la classe à laquelle appartient l'objet. L'objectif de la carte de saillance est de trouver les régions qui sont proéminentes ou perceptibles à chaque endroit du champ.

3.3.1 Algorithme de la carte de saillance

Algorithme : Carte de saillance

Data : un graphe connexe $G = (V, E)$, un arbre T d'une hiérarchie H de V, et un array de niveaux qui associe à chaque noeud de T sa hauteur (qui est également le niveau auquel la région correspondate apparaît pour la première fois dans la hiérarchie)

Result : La carte de saillance $S = \phi_G(\mathcal{H})$ de la hiérarchie $\mathcal{H}$.

```
1 LCAPreprocess(T);
2 foreach arête {x, y} ∈ E do
3 |   S[{x, y}] := level[LCA(T, {x}, {y})] − 1;
4 end
```

où *Lowest Common Ancestor* (LCA) est le plus petit ancêtre commun entre les deux sommets x et y dans l'arbre T. Pour plus de détails concernant cet algorithme voir la section 7 du document J. Cousty et al.[2].

Voici une illustration d'une carte de saillance. La carte (représentée par des chiffres noirs) est la carte de saillance $S = \phi_G(\mathcal{H})$ de la hiérarchie H illustrée à la Figure 3.1 lorsque nous considérons le graphe G représenté en gris[2] :

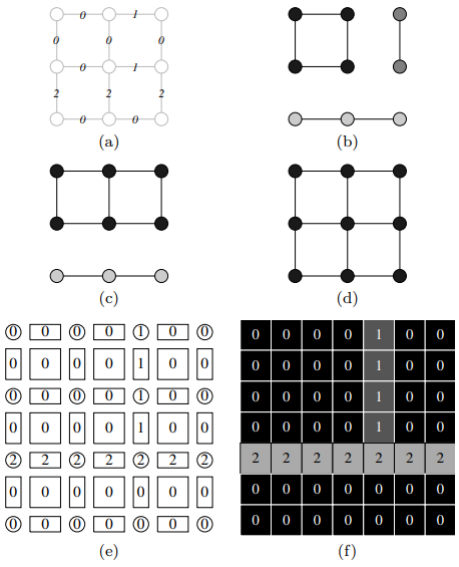


FIGURE 3.3 – Illustration d'une carte de saillance

— (b, c, d) les graphes à 1, 2 et 3 niveaux de G pour s. Les sommets sont colorés en fonction des partitions à 1, 2 et 3 niveaux de G : dans chaque sous-figure,

deux sommets appartenant à une même composante connectée ont le même niveau de gris.

— Les sous-figures (e) et (f) montrent les représentations possibles d'une carte de saillance lorsque l'on considère le graphe de 4-adjacences.

3.4 Algorithme Watershed

Afin de calculer une ligne de partage des eaux hiérarchique, une idée clé des algorithmes de [12,35] en référence de la publication de J. Cousty et al.[2] est de calculer une carte de poids dont la hiérarchie des zones quasi-plates correspond à la segmentation hiérarchique souhaitée de la ligne de partage des eaux. Cela permet de réduire la complexité temporelle par rapport à un calcul direct de la hiérarchie. Ainsi l'algorithme présenté prend avantage sur les liens entre carte de saillance et hiérarchie. Voici les différentes étapes de l'algorithme :

Algorithme 2 *Algorithme Watershed*

1. Étant donné le graphe pondéré (G, w), il faut premièrement calculer un arbre de partition binaire par ordre d'altitude que l'on nommera *Binary Partition Tree by Altitude Ordering* (BPTAO). Cette structure est la hiérarchie des partitions de V obtenue lors de l'algorithme de l'arbre minimum de Kruskal. Nous considérons d'abord une partition en singletons. Puis, lorsqu'une arête est sélectionnée par l'algorithme de Kruskal, nous construisons le niveau suivant de la hiérarchie en fusionnant les plus grandes régions contenant les sommets de l'arête sélectionnée. En termes d'arbre, la région nouvellement créée est un nouveau nœud de la BPTAO, qui devient le parent des deux nœuds associés aux régions fusionnées. Ainsi, la BPTAO obtenue est un arbre dont les nœuds (mais pas les feuilles) correspondent aux arêtes de l'arbre minimal T produit par l'algorithme de Kruskal.
2. Avec la BPTAO obtenue, les minima de la carte de poids sont identifiés et les attributs régionaux ainsi que les valeurs d'extinction des minima peuvent être calculés. Pour calculer l'attribut régional il faut alors parcourir la BPTAO une première fois, des feuilles à la racine, et une deuxième traversée de la racine aux feuilles permet d'obtenir les valeurs d'extinction.
3. Une fois les valeurs d'extinction des minima obtenues, elles peuvent être étendues à tous les nœuds de l'arbre : l'extinction d'un nœud non feuille est la valeur d'extinction la plus élevée de ses descendants. Ces valeurs peuvent être calculées en parcourant à nouveau l'arbre des feuilles à la racine. Aux étapes 2 et 3, nous ne travaillons que sur la représentation arborescente directe de la hiérarchie initiale.

4. Puis, nous fixons la persistance de chaque nœud (hors feuille) au minimum de l'extinction de ses deux enfants. Ainsi, nous obtenons une persistance pour chaque nœud non feuille de la BPTAO. Étant donné que les nœuds (hors feuilles) de la BPTAO correspondent aux arêtes du MST, nous obtenons une valeur de persistance pour chaque arête de l'arbre minimal. En d'autres termes, nous avons produit une nouvelle carte de poids p (par les valeurs de persistance) pour les arêtes du MST qui est T.
5. La hiérarchie de watershed est donc la hiérarchie des zones quasi-plates de T pour la carte p obtenue. Aux étapes 4 et 5, la nouvelle hiérarchie de watershed est construite en considérant d'abord sa carte de saillance (étape 4) avant de calculer explicitement la hiérarchie (étape 5). Ainsi, à ces étapes, nous tirons parti des liens entre les cartes de saillance et les hiérarchies établis par les théorèmes 1 et 2 définit par J. Cousty et al[2].

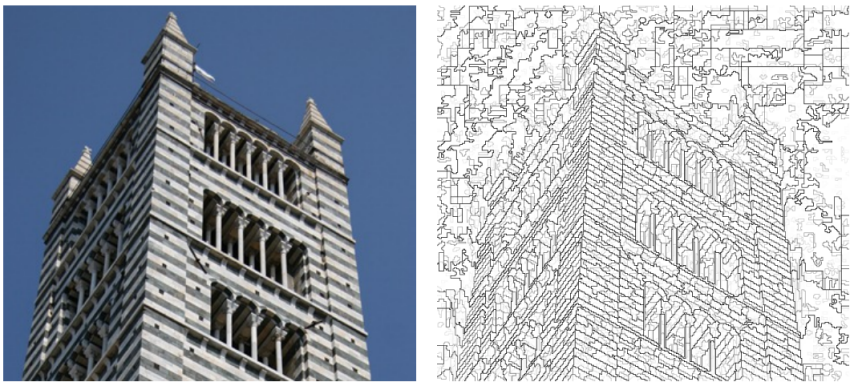


FIGURE 3.4 – Exemple d'une hiérarchie de watershed (carte de saillance) à partir d'une image

Chapitre 4

Implémentation de la méthode Felzenszwalb et Huttenlocher

L'implémentation de la méthode de Felzenszwalb et Huttenlocher est inspirée de Ghassem Alaee : github.com/salaee/pegbis. Celle-ci reprend les principes décrits dans ce document, et donc de ceux de Felzenszwalb et Huttenlocher dans leur publication[1].

4.1 Fonction segment_felzenszwalb

La fonction principale est $segment\_felzenszwalb$ qui prend en entrée une image de dimensions $(width, height, 3)$ respectivement largeur, hauteur et nombre de canaux pour les couleurs RVB qui sont obligatoirement 3 ici. Il faut également définir les termes *sigma* qui sert au filtrage gaussien appliqué à l'image pour atténuer l'image sur les 3 canaux RVB, la constante k qui sert dans notre fonction de seuil (voir section 2.2.4) ainsi que min_size qui définit une taille minimale pour les composantes. Voici le déroulé de la fonction :

1. Ainsi dans un premier temps nous appliquons un filtre gaussien sur les trois canaux RVB de l'image pour atténuer les changements brusques de teintes de pixels.

2. Puis nous bouclons sur tous les pixels de l'image pour construire le graphe pondéré G des pixels avec leur voisinages, où chaque sommets est un pixel. Puis nous bouclons sur tous les pixels (x, y) de l'image pour construire le graphe pondéré des pixels avec leur voisinages, où chaque sommets est un pixel. Si cela est possible pour chaque sommet $(x, y) \in V = \{(0, 0), ..., (width - 1, height - 1)\}$:
   - Si $x < width - 1$, ajouter une arête avec le sommet $(x + 1, y)$ de poids $w = \sqrt{(R_{x,y} - R_{x+1,y})^2 + (V_{x,y} - V_{x+1,y})^2 + (B_{x,y} - B_{x+1,y})^2}$
   - Si $y < height - 1$, ajouter une arête avec le sommet $(x, y + 1)$ de poids $w = \sqrt{(R_{x,y} - R_{x,y+1})^2 + (V_{x,y} - V_{x,y+1})^2 + (B_{x,y} - B_{x,y+1})^2}$
   - Si $x < width - 1$ et $y < height - 2$, ajouter une arête avec le sommet $(x + 1, y + 1)$ de poids $w = \sqrt{(R_{x,y} - R_{x+1,y+1})^2 + (V_{x,y} - V_{x+1,y+1})^2 + (B_{x,y} - B_{x+1,y+1})^2}$
   - Si $x < width-1$ et $y > 0$, ajouter une arête avec le sommet $(x+1, y-1)$ de poids $w = \sqrt{(R_{x,y} - R_{x+1,y-1})^2 + (V_{x,y} - V_{x+1,y-1})^2 + (B_{x,y} - B_{x+1,y-1})^2}$

   Ainsi les poids des arêtes est égale à la valeur de différence d'intensité sur les 3 canaux RVB. Un exemple des cas de ce 4-voisinages est montré en Figure **??**, les pixels gris sur la figure sont cas possibles et les pixels verts autours le 4-voisinages exprimés ci-dessus.
3. Puis à l'aide de la fonction *segment_graph*, on construit la structure de données à ensemble disjoint Union-find (représenté par la classe $Universe$) où au départ chaque pixel est une composante unique. On boucle sur les arêtes de G dans l'ordre croissant des poids entre les paires de sommets (v_i, v_j) :
   (a) On utilise $find$ sur la structure Union-find pour récupérer la composante à laquelle appartient le sommet v_i et v_j, que l'on nommera ici C_i et C_j
   (b) Si $C_i \neq C_j$, $w(v_i, v_j) \leq \tau(C_i)$ et $w(v_i, v_j) \leq \tau(C_j)$ alors fusionner C_i et C_j
4. A partir de la segmentation établie ci-dessus (dans l'objet $Universe$), nous bouclons de nouveau sur les arêtes pour vérifier que la taille des composantes formées soient bien supérieur à la constante *min_size* auquel cas on fusionne alors les paires de composantes des paires de sommets des arêtes parcourues.
5. Finalement, il ne nous reste plus qu'à attribuer des couleurs uniques aléatoirement pour chaque composante de la segmentation obtenue.

Nous obtenons donc dans le tableau *output* de dimensions égale à l'image où chaque valeur aux coordonnées (x, y) est la couleur de la composante à laquelle le pixel (x, y) est attribué par la segmentation produite.

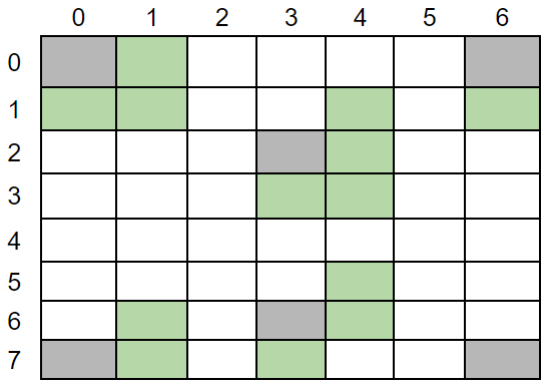


FIGURE 4.1 – Exemple du 4-voisinages sur une image 7x8

Chapitre 5

Implémentation de la méthode watershed

Pour l'implémentation de la méthode par watershed, nous avons utilisés la librairie *higra* qui nous fourni notamment un notebook (higra.readthedocs.io) qui montre un exemple avec la librairie pour effectuer une segmentation par watershed avec la méthode présentée dans le document de J. Cousty et al[2].

5.1 Fonction segment_watershed

La fonction principale pour cette méthode est *segment_watershed*, celle-ci prend en entrée les données de l'image, les dimensions de l'image et le nombre de régions avec la plus grande cardinalité à retenir après segmentation nommé *n_comp*. Voici le déroulé de la fonction :

1. On calcule le gradient de l'image par détection des contours à l'aide de la fonction *detectEdges* de l'objet créé $cv2.ximgproc.createStructuredEdgeDetection(get\_sed\_model_file())$. Cet objet permet la détection des contours d'une image par le gradient en utilisant un modèle de machine learning entraîné pour cette tâche. Ce qui permet une meilleure qualité de détection.

2. Puis à partir de l'image, on construit un graphe d'adjacence non orienté G de 4-voisinages des pixels avec la fonction *higra.get_4_adjacency_graph*
3. On transfert les valeurs du gradient de l'image calculé aux poids des arêtes, ce qui pondéré le graphe G.
4. A partir du graphe pondéré G et de ces arêtes avec la méthode *higra.watershed_hierarchy_by_area*, nous pouvons ainsi obtenir une hiérarchie de watershed. Cette fonction retourne l'arbre correspondant et les altitudes de ses noeuds.
5. Ce qui nous permet à partir de l'arbre et les altitudes des noeuds de pouvoir obtenir le graphe de la carte de saillance avec la fonction *higra.saliency*.
6. Nous filtrons le graphe de la carte de saillence pour pouvoir ne retenir exactement les *n_comp* régions les plus grandes.
7. Enfin à partir du graphe pondéré G et du graphe de la carte de saillance, en utilisant la fonction *higra.labelisation_watershed*, nous obtenons le tableau de même dimensions que l'image, où chaque valeur de coordonnées (x, y) est l'attribution de chaque pixel à une composante. Comme nous avons filtrés en ne gardant que les *n_comp* régions les plus grandes, tous les pixels qui n'étaient pas dans une des ces *n_comp* régions se retrouvent tous dans une composante de même id.

Chapitre 6

Expérience menée

Pour pouvoir comparer la qualité de segmentation des deux méthodes, nous avons implémentés le calcul des bounding box à partir des segmentations obtenues, et comparer celles-ci avec les groundtruth des images. Nous avons effectuer cette expérience sur 10 images de quatres catégories différentes. Les sections suivantes relatent l'implémentation, résultats et analyses de cette expérience.

6.1 Dépôt github

L'entièreté du projet est disponible en ligne sur github : github.com/patherstevenson/M1-PJI

6.2 Dataset utilisé

Pour mêner cette expérience, nous avons fait le choix de prendre le jeu de données **Pascal VOC 2012**, qui est le jeu de données standard pour la segmentation, la détection et la localisation d'images, etc. Celui-ci en disponible en ligne en téléchargement sur kaggle.com/datasets

6.3 Implémentation des bounding box

Pour implémenter les bounding box, nous avons fait le choix de faire une classe *BndBox* qui prend en paramètre *label* pour un nom de catégorie d'objets à détecter

(par exemple : person,chair,bird,car, etc...) et les dimensions w et h de l'image pour laquelle cet objet calculera la bounding box.

6.3.1 Calcul des bounding box

Ainsi, nous avons ajouter la déclaration de cet objet en toute fin des fonctions $segment\_felzenszwalb$ (4.1) et $segment\_watershed$ (5.1). Puis en itérant sur chaque pixel, nous utilisons la méthode $BndBox.check\_pixel$ pour savoir si un pixel est le pixel est le pixel le plus à gauche et/ou le pixel le plus à droite et/ou le pixel le plus haut et/ou le pixel le plus en bas de la composante à laquelle il appartient. Nous enregistrons donc ces informations pour chaque composante dans un attribut dictionnaire de la classe $BndBox.bndbox$. A partir de ces points il est donc possible de calculer la dimension des bounding box associés à chaque composante.

6.4 Evaluation de la qualité des bounding box

L'évalution des bounding box est inspiré de la méthode de mesure de qualité de la publication de Uijlings, Jasper and Sande, K. and Gevers, T. and Smeulders, A.W.M.[3] dans la reconnaissance d'objets.

6.4.1 Calcul de l'overlapping entre deux rectangles

Pour calculer l'overlap entre deux rectangles nous utilisons la méthode $BndBox.overlap$ qui prend en argument l_1, r_1, l_2, r_2 :

- l_1 : le point de coordonées du coin bas gauche du premier rectangle
- r_1 : le point de coordonées du coin hauth droite du premier rectangle
- l_2 : le point de coordonées du coin bas gauche du second rectangle
- r_2 : le point de coordonées du coin hauth droite du second rectangle

Puis à partir de ceux-ci nous calculons l'overlap entre les deux rectangles de la façon suivante :

$$\begin{aligned} area_1 &= |(l1_x - r1_x)| \times |(l1_y - r1_y)| \\ area_2 &= |(l2_x - r2_x)| \times |(l2_y - r2_y)| \end{aligned} \tag{6.1}$$

$$\begin{aligned} x_{dist} &= \min(r1_x, r2_x) - \max(l1_x, l2_x) \\ y_{dist} &= \min(l1_y, l2_y) - \max(r1_y, r2_y) \end{aligned} \tag{6.2}$$

$$area_I = \begin{cases} x_{dist} \times y_{dist} & \text{si } x_{dist} > 0 \text{ et } y_{dist} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{6.3}$$

$$Overlap = \frac{area_I}{(area_1 + area_2 - area_I)} \tag{6.4}$$

6.4.2 Calcul de l'Average Best Overlap

Pour évaluer la qualité de nos hypothèses d'objets, nous définissons les scores *Average Best Overlap* (ABO) et *Mean Average Best Overlap* (MABO). Pour calculer le meilleur recouvrement moyen pour une classe spécifique c, nous calculons le meilleur recouvrement entre chaque annotation de vérité terrain $g_i^c \in G^c$ et les hypothèses d'objet L générées pour l'image correspondante, et nous faisons la moyenne[2] :

$$ABO = \frac{1}{|G^c|} \sum_{g_i^c \in G^c} \max_{l_j \in L} Overlap(g_i^c, l_j) \tag{6.5}$$

Dans notre implémentation cette évalution est orchestrée en utilisant deux méthodes dans l'ordre, *BndBox.init_eval* et *BndBox.start_eval*.

6.4.3 *BndBox.init_eval*

Cette méthode qui prend en argument *gt_path* comme chemin vers le fichier des vérités terrain associé à l'image et *category* pour le nom de catégorie des objets à détecter dans l'image, l'appel à cette méthode initialise les attributs suivants :

- df_bndbox : un DataFrame pandas contenant les coordonées des rectangles des vérités terrain des objets de la catégorie *category*
- overlap_05 : dictionnaire pour stocker la bounding box de chaque vérité terrain qui aura le plus grand overlap au moins supérieur strictement à 0.5
- max_overlap : dictionnaire pour stocker la bounding box de chaque vérité terrain qui aura le plus grand overlap
- abo : dictionnaire pour stocker les scores ABO pour chaque vérité terrain de la catégorie
- bndbox_color : dictionnaire pour stocker les couleurs de chaque bounding box (rouge ou vert)

6.4.4 *BndBox.start_eval*

Une fois que la méthode *BndBox.init_eval* a initialisé toutes les structures nécessaire, la méthode *BndBox.start_eval* va lancer les comparaisons entre les vérités terrain est les bounding box précédemment calculer et stocker dans *BndBox.bndbox*.

Et permettre de remplir les dictionnaires nécessaire pour calculer l'ABO et MABO.

1. Ainsi pour chaque vérité terrain du présente dans le DataFrame *df_bndbox* :
   (a) récupérer les points l_1, r_1 de la vérité terrain actuellement parcouru
   (b) pour chaque bounding box dans *BndBox.bndbox* :
      i. récupérer les points l_2, r_2 de la bounding box actuellement parcouru
      ii. calculer l'overlap avec la méthode $BndBox.overlap(l_1, r_1, l_2, r_2)$
      iii. Si l'overlap > 0.5 et à l'overlap stocké dans *overlap_05* pour cette vérité terrain alors :
         A. attribué la couleur rouge à la bounding box stocké dans *overlap_05* pour cette vérité terrain
         B. mettre à jour *overlap_05* pour cette verité terrain avec le nouvel meilleur overlap obtenu (> 0.5)
         C. attribué la couleur verte à cette bounding box
      iv. Si l'overlap calculé est supérieur à celui stocké dans *max_overlap* pour cette vérité terrain alors mettre à jour *max_overlap* avec cette bounding box et cet overlap
2. Enfin nous lançons la méthode *BndBox._eval_abo* qui lançera le calcul du MABO conformément à l'équation 6.5.

6.5 Fonction principale : *segmentation*

Pour permettre de n'utiliser qu'une seule fonction pour notre expérience, nous définisons la fonction *segmentation*. Celle-ci prend en argument :

— *input_path* : chemin vers le fichier image à segmenter
— *method* : méthode de segmentation souhaitée (felzenszwalb ou watershed)
— *kwargs* : dictionnaire de paramètre pour la méthode felzenszwalb (*sigma*, *k*, *min_size*)
— *n_comp* : nombre des plus grandes régions à retenir dans la hiérarchie de watershed
— *category* : nom de la catégorie des objets à détecter
— gt_path : chemin vers le fichier xml de groundtruth, cet argument facultatif si vous respectez l'arborescence et le nommage de fichier fourni dans l'implémentation.
— *save* : booléen pour indiquer si l'on souhaite sauvegarder l'image de la segmentation obtenue

— *verbose* : booléen pour la verbosité

Au niveau de l'exécution de cette fonction, celle-ci effectue les étapes suivantes :

1. Récupérer les données de l'image en entrée à partir du chemin image donné.
2. Récupérer les dimensions de l'image.
3. Appeler $segment\_felzenszwalb$ ou $segment\_watershed$ avec les bons arguments de la méthode indiquée en paramètre, en retourne de cet appel on récupéré un tuple $(output, bb)$ contenant $output$ l'image segmentée avec la coloration aléatoire appliquée et bb l'objet $BndBox$ contenant les bounding box calculées en fin des fonctions $segment\_felzenszwalb$ et $segment\_watershed$.
4. Si aucun chemin vers un fichier xml est indiqué pour les vérités terrain dans gt_path, alors récupérer le chemin respectant l'arborescence du dataset $VOC2012$ associé à l'image donnée en entrée.
5. Initialisé des structures nécessaire à l'évalutiondes bounding box : appel de la méthode $BndBox.init\_eval$ avec pour argument le chemin vers le fichier xml des vérités terrain et le nom de catégorie des objets à détecter dans l'image.
6. Lancement des évalutions des bounding box avec l'appel de la méthode $BndBox.start\_eval$, l'argument de verbosité est défini à $False$ par défaut car celui-ci affiche toutes les comparaisons d'overlap effectuées.
7. Puis nous utilisons la fonction $plot\_segment$ en indiquant bien la constante k donnée en paramètre pour obtenir l'affichage de l'image les bounding box rouges et vertes obtenues après évalution et l'image segmentée. Cet affichage est sauvegardé dans le dossier $result$.

6.6 L'expérience

Ainsi nous avons tous les éléments pour exécuter convenablement notre expérience de comparaison entre les deux méthodes sur plusieurs catégories d'objets. Nous avons utiliser pour notre expérience un jupyter notebook disponible sur le dépôt git : src/main.ipynb.

Ainsi nous utiliserons 10 images pour 4 catégories différentes à savoir *cat*, *person*, *chair* et *bicycle*. Pour chaque catégorie nous récupérons les chemins des images et pour chaque image nous effecutons la segmention avec les deux méthodes pour chaque constante k de fonction de seuil $k = [50, 150, 250, 350, 450, 550, 650, 750, 850, 950, 1050]$. Nous effectuons la segmentation avec la méthode de Felzenszwalb en premier pour ainsi connaître combien de régions sont obtenues pour le k actuel et ainsi pouvoir donner ce nombre comme paramètre n_comp étant donner que nous

n'avons pas l'information en avance du nombre de régions obtenues pour un k donné avec la méthode Felzenszwalb. Ce qui est nécessaire pour pouvoir comparer les segmentations des deux méthodes en ayant autant de régions retenues avec watershed que de régions obtenues avec Felzenszwalb. On précise également le k en argument lors de la fonction *segmentation* pour la segmentation avec watershed de manière à avoir le k dans le nom du plot du résultat (voir fonction *plot_segment*), ce qui permet de pouvoir également comparer les deux segmentations d'une image à une même étape k dans les fichiers de résultats. Enfin nous récupérons les MABO pour chaque segmentation dans des dictionnaires et effectuons la moyenne pour chaque k par catégorie sur les 10 images. Ce qui nous permettra d'afficher les résultats des MABO en fonction de la constante de seuil k. Toutes les segmentations sont stockées et organiser par catégorie dans le dossier *result/* du dépôt git.

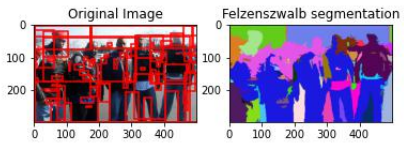


FIGURE 6.1 – Exemple d'une segmentation par Felzenszwalb

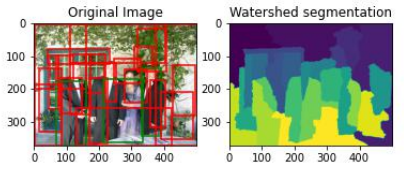
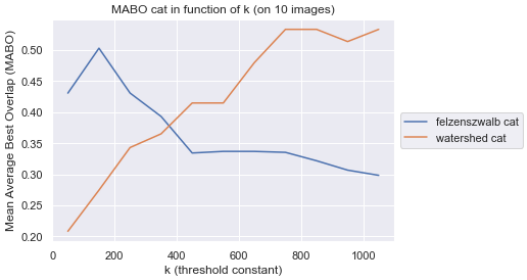


FIGURE 6.2 – Exemple d'une segmentation par watershed

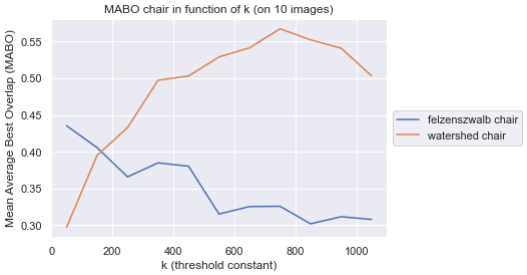
6.7 Résultats

Ainsi à l'aide des MABO calculés, nous pouvons effectuer la moyenne pour chaque k par catégorie. Voici ce que nous obtenons pour les 4 catégories étudiées :



(a) MABO sur 10 images pour la catégorie *cat*

(b) MABO sur 10 images pour la catégorie *person*



(c) MABO sur 10 images pour la catégorie *cat*

(d) MABO sur 10 images pour la catégorie *person*

FIGURE 6.3 – Moyenne du MABO par k sur 10 images par catégorie

On peut constater qu'un patterne ressort entre les deux méthodes par rapport à la constante de seuil k. En rappel qu'au plus k augmente au plus grande les régions sont grandes, donc au moins de régions nous avons. Pour la première méthode de Felzenszwalb plus k augmente plus la qualité de la détection diminue, alors que pour la seconde méthode par watershed au plus k augmente au plus la qualité de la segmentation est bonne, avec une certaine stagnation de celle-ci pour certaine catégorie.

On observe globalement au travers de cette expérience que très majoritairement la segmentation par watershed permet d'obtenir de meilleure segmentation pour la détection de catégorie d'objets dans les images si l'on arrive à définir la constante de fonction de seuil k suffisamment grande en fonction de la catégorie d'objet. Tout en sachant que l'obtention du gradient des images par la librairie *higra* utilise un modèle pré-entraîné de machine learning, ce qui permet sûrement d'obtenir une meilleure segmentation en finalité comparer à la méthode de Felzenszwalb.

Chapitre 7

Conclusion

Au travers de ce projet nous avons étudier et implémenter deux méthodes significative de la segmentation d'image et ainsi observer des résultats intéressants en menant une expérience. Notamment l'importance de la constante de seuil k dans le cas de la méthode de Felzenszwalb et Huttenlocher, et du nombre de plus grandes régions à conserver dans la méthode par watershed pour obtenir le maximum en terme de qualité de segmentation dans l'évalution des bounding box. Nos résultats permettent de conclure sur la supériorité de la méthode par watershed pour obtenir des bouding box les plus proches des vérités terrain. Même si pour certaines catégories l'écart entre les deux méthodes n'est pas si significatif. Nos conclusions sont donc à considérer que pour l'expérience mênée, une expérience de plus grande envergure en terme d'images par catégorie et de nombre de catégorie pourrait permettre de venir conclure de manière concrète sur nos observations.

7.1 Perspectives

Dans un soucis de temps au niveau de ce projet il n'était pas permis de conduire l'intégration des méthodes de segmentation d'images implémentées à un modèle R-CNN de détection d'objets par méthodes d'apprentissage profond[4]. C'est une suite intéressante qui peut être donnée à ce projet au vue des avancées constantes dans la détection d'objets par apprentissage machine. Une autre question de recherche pertinente pourrait être de trouver une méthode pour connaître avant segmentation

le nombre de régions obtenues après segmentation en fonction de la constante de la fonction de seuil k pour les méthodes implémentées.

Références

[1] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient graph-based image segmentation," *International Journal of Computer Vision*, vol. 59, pp. 167–181, Sep 2004. [Cité en pages 3, 4, 5, 8 et 19]

[2] J. Cousty, L. Najman, Y. Kenmochi, and S. Guimarães, "Hierarchical segmentations with graphs : Quasi-flat zones, minimum spanning trees, and saliency maps," *Journal of Mathematical Imaging and Vision*, vol. 60, pp. 479–502, May 2018. [Cité en pages 11, 12, 13, 14, 15, 16, 17, 23 et 27]

[3] J. Uijlings, K. Sande, T. Gevers, and A. Smeulders, "Selective search for object recognition," *International Journal of Computer Vision*, vol. 104, pp. 154–171, 09 2013. [Cité en page 26]

[4] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014 IEEE Conference on Computer Vision and Pattern Recognition, pp. 580–587, 2014. [Cité en page 33]