GREYC
Laboratoire de recherche en sciences du numérique

M2 Informatique
IDM
Année 2021-2022

UNIVERSITÉ
CAEN
NORMANDIE

Apprentissage profond de superpixels

Rapport de Stage

Etudiant :
DALLOCCHIO François

Encadrants :
BOUGLEUX Sébastien
LÉZORAY Olivier

Table des matières

1 Introduction **2**
1.1 Entreprise d'accueil 2
1.2 Problématique et enjeux 2

2 SLIC **4**
2.1 fonctionnement de l'algorithme 4
2.2 Utilité du postprocessing 6
2.3 Comparaison par rapport aux autres algorithmes de superpixels . . . 6

3 Principe de la recherche **8**

4 Extraction des features **9**
4.1 Composition des réseaux convolutif 9
4.2 Le hooking 10

5 Implémentation **11**
5.1 implémentation de SLIC 11
5.2 implémentation de l'extraction des features 13
5.3 ajout des features à SLIC 14

6 Expérimentation **15**
6.1 Benchmarks 15
6.2 Protocole expérimental 15
6.3 Résultats et analyse 17

Conclusion **24**

Références **25**

1 Introduction

Le traitement de Données est utilisé dans de nombreux domaines actuels tels que le domaine médical ou le domaine automobile avec les voitures autonomes. En effet, avec l'augmentation du volume de ces données, les temps de traitement deviennent de plus en plus long et peuvent être contraignants pour des calculs embarqués. Des méthodes de pré-traitement ont été inventées pour permettre d'avoir des résultats plus rapides. L'expansion du Deep Learning a également permis de développer de nouvelles manières pour traiter des données.

1.1 Entreprise d'accueil

J'ai réalisé mon stage au sein du GREYC, le Groupe de recherche en informatique, image, automatique et instrumentation de Caen, travaillant dans l'équipe Image. Ce laboratoire de recherche est composé de plusieurs équipes de recherche, menant différentes activités autour de thèmes précis. Ces équipes sont :

- l'équipe **AMACC** (Algorithmes, Modèles de calcul, Aléa, Combinatoire, Complexité) : Recherche en informatique mathématique et modèle de calcul.
- l'équipe **CODAG** (Contraintes, Ontologies, Données, Annotations, Graphes) : Fouille de données, contraintes et graphes.
- l'équipe **Electronique** : Étude des capteurs à haute et à faible sensibilité.
- l'équipe **Image** : Traitement et analyse de signaux, images et vidéos.
- l'équipe **MAD** (Modèles, Agents, Décision) : Agents autonomes et systèmes multi-agents.
- l'équipe **SAFE** (Sécurité, Architecture, Forensique, biomEtrie) : Recherche en Sécurité informatique à travers la biométrie, l'architecture et modèles de sécurité, la forensique.

Ces équipes travaillent donc autour d'axes majeurs qui sont la science des données, les capteurs et instruments et les algorithmes et intelligences artificielles. Les recherches du laboratoire peuvent permettre de trouver des réponses à des problèmes sur le thème de la santé mais également sur les sciences humaines et sociales.

Je souhaite remercier l'aglomération de Saint-Lô qui a financé ce stage, et l'équipe Image du GREYC de m'avoir accueilli.

Enfin, je tiens tout particulièrement à remercier mes deux encadrants de stage, Mr Sébastien BOUGLEUX et Mr Olivier LÉZORAY, qui se sont impliqués et m'ont aidés tout au long de ce stage, ce qui m'a permis de mener à bien notre recherche.

1.2 Problématique et enjeux

Ces dernières années, on a pu assister à une augmentation de l'adoption de l'apprentissage profond pour un large éventail de problèmes de traitement et analyse

d'images. Les superpixels sont cependant rarement utilisés en conjonction avec les réseaux profonds.

Un superpixel est un groupe de pixels qui partage des caractéristiques similaires, comme la couleur, ou l'intensité par exemple. Chaque pixel appartenant à un même superpixel contient le même label. Ils servent dans différents domaines de la segmentation d'image comme la détection d'objet ou la labellisation. Un superpixel contient donc plus d'informations qu'un seul pixel, sa forme peut aussi varier pour suivre les contours d'une image alors qu'un pixel a une forme et taille fixe. Une image segmentée en superpixels est utile pour la réalisation de problèmes informatiques coûteux car elle va diminuer le temps de calcul dû principalement à la différence de nombre entre une image de pixels de un image de superpixels.

L'utilisation de l'apprentissage profond pour les superpixels appliquée à des algorithmes de segmentation classiques tels que SLIC, qui opère uniquement sur des caractéristiques de couleur, permettrait de meilleurs résultats sur des domaines d'images plus spécifiques comme le domaine medical.

Notre problématique naît de cette hypothèse :

Peut-on obtenir de meilleurs résultats pour SLIC en utilisant des caractéristiques issues de réseaux profonds ?

Nous allons expliquer comment fonctionne l'algorithme SLIC et comment nous l'avons utilisé pour nos expérimentations. Nous évoquerons aussi l'extraction de caractéristiques provenant de réseaux profonds qui ont été utiles pour implémenter notre propre algorithme de SLIC modifié. Nous montrerons enfin les résultats apportés à nos essais via différentes mesures et graphiques pour déterminer si nos recherches ont des résultats positifs.

2 SLIC

Pour obtenir une image de superpixels, plusieurs algorithmes ont été développés, souvent dérivés d'algorithmes de segmentation ou de classification. Les algorithmes sont divisés en 2 approches car les méthodes utilisées par ceux ci sont basés soit sur des méthodes de graphes, où un pixel est considéré comme un noeud dans un graphe, soit sur des méthodes de gradient, où les superpixels sont modifiés à chaque itération jusqu'à un seuil de convergence.

Nous retrouvons donc des algorithmes utilisant la croissance de région comme la technique de montée des eaux, ou encore d'autres utilisant des coupes de graphes.

2.1 fonctionnement de l'algorithme

L'algorithme SLIC (Simple Linear Iterative Clustering) est un algorithme de gradient, imaginé en 2010 par Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, et Sabine Süsstrunk. Il a été conçu dans le but de générer des superpixels, contrairement à d'autres algorithmes où ceci n'est pas le but premier. L'approche de SLIC est de créer des superpixels en fonction de la similarité de la couleur et de la proximité des pixels d'une image. Cela se fait dans un espace 5D [labxy], avec les coordonnées xy et l'espace de couleur CIELAB, où L correspond à la clarté, a représente la valeur sur un axe vert $\rightarrow$ rouge et b la valeur sur un axe bleu $\rightarrow$ jaune. A chaque itération, le cluster change de forme pour avoir une meilleure segmentation jusqu'à convergence. Voici l'algorithme de base en 2010 :

Algorithm 1: SLIC SUPERPIXEL SEGMENTATION

```
/* Initialisation */
Initialiser les centres de cluster C_k=[l_k, a_k, b_k, x_k, y_k]^T en prélevant des pixels
  sur une grille de pas régulier S
Déplacer les centre à la position du plus petit gradient dans un voisinage 3x3
Définir un tableau de label l(i)=-1 pour chaque pixel i
Définir un tableau de distance d(i)=∞ pour chaque pixel i
répéter
/* Assignement */
for Chaque centre de cluster C_k do
    for Chaque pixel i dans une région 2Sx2S autour de C_k do
        Calculer la distance D entre C_k et i.
        if D < d(i) then
            d(i)=D
            l(i)=k
/* Actualisation */
Calculer les nouveaux centres
Calculer l'erreur résiduelle E
jusqu'à E ≤ seuil
/* Post traitement */
```

Il n'y a qu'un seule entrée, K le nombre désiré de superpixels. Nous avons N qui est le nombre de pixels de l'image. Ces deux valeurs servent à définir l'intervalle S qui sert à l'initialisation des centres et la taille de la région pour le calcul de distance des centres. La relation entre ces trois valeurs est $S = \sqrt{N/k}$. Pour D, nous calculons plusieurs distances : $Dlab$, distance entre les vecteurs couleur ans l'espace CIELAB, et Dxy distance entre les vecteurs 2D des coordonnées despoints, D est l'association des deux mesures et équivaut à $D = Dlab + m/S \times Dxy$ avec m la compacité du superpixel(généralement entre 1 et 20). Quand m est petit,les superpixels collent mieux aux contours mais leurs formes sont moins régulières, quand m est grand, les superpixels sont plus compacts. Les pixels vont s'associer au centre où la mesure Ds est la plus petite.

FIGURE 1 – Image de base

FIGURE 2 – Image segmentée avec m=20

FIGURE 3 – Image segmentée avec m=41

On peut voir sur les figures 2 et 3 qu'il faut choisir m assez grand pour avoir des superpixels de formes régulières.

Les K centres de clusters vont se déplacer à la position du plus petit gradient dans un voisinage 3x3. cela empêche de les placer dans un coin. les pixels de l'images s'associent aux clusters dont le centre est le plus proche, puis le centre est recalculé en fonction des points du cluster. On boucle jusqu'à convergence E qui est l'erreur residuelle ; la distance entre l'ancien et le nouveau centre. Une phase de post traitement est présente en fin d'algorithme pour corriger des erreurs de segmentation.

L'algorithme est un dérivé de K-means, un algorithme de clustering où des centres sont placés aléatoirement sur une image, les points les plus proches sont associés à leur centre respectif, le centre se déplace au milieux des points associés, puis les points sont réassignés au centre le plus proche, jusqu'à avoir des clusters stables. Les différences majeures entre SLIC et K-means sont que pour SLIC, les centres initiaux ne sont pas générés aléatoirement, mais sont placés en forme de grille et en fonction du nombre de superpixels que nous souhaitons, et que l'assignation se fait sur une zone de taille délimitée de deux fois la distance initiale entre deux centres).

A chaque itération, chaque centre n'a donc pas besoin d'un traitement complet de chaque pixel de l'image ce qui améliore grandement sa vitesse de calcul. La complexité de SLIC est donc linéaire en fonction du nombre de pixels de l'image et est O(N), alors que les algorithmes de K-moyennes sont O(kNI) avec I le nombre d'itérations.

2.2 Utilité du postprocessing

Pour fonctionner l'algorithme contient une phase de post traitement après la boucle principale de l'algorithme. Elle est très utile car elle permet de corriger les plus gros défaut de clustering laissés par l'algorithme. Ici, le but du post processing est d'affecter des petites zones d'un morceau de superpixel isolé au superpixel le plus proche. Sans cette manipulation, les résultats de SLIC peuvent être mauvais surtout sur des images complexes.

FIGURE 4 – Image segmentée avec SLIC sans post traitement

FIGURE 5 – Image segmentée avec SLIC avec post traitement

Nous pouvons voir sur les Figures 4 et 5 deux images segmentées par SLIC. La première, non post-traitée, possède plusieurs petits défauts de segmentation qui sont corrigés grâce à un post traitement comme visible sur la seconde image. L'objectif du post-traitement est donc le suivant : toutes les petites zones qui n'ont pas été rattachées au superpixel qui les contiennent vont être attribués dans leur superpixel respectif le plus proche.

Sans post traitement, l'efficacité de SLIC pourrait être assez décevante. C'est pourquoi le post traitement est primordial à la qualité de segmentation de SLIC. Il serait donc intéressant de montrer lors de calculs de benchmarks à quel point le post traitement impacte les résultats de SLIC.

2.3 Comparaison par rapport aux autres algorithmes de superpixels

L'équipe qui a développé SLIC a également voulu comparer son algorithme par rapport aux autres méthodes générant des superpixels. Un des avantages de SLIC est qu'il est facile à paramétrer, car il n'y en a qu'un seul, le nombre de superpixels.

Nous avons également montré un défaut de SLIC par rapport à d'autres méthodes de segmentation ; la nécessité de post traitement. En effet, sans celui ci, les résultats de SLIC sont moins bons, alors que des algorithmes de croissance de régions notamment permettent de créer des superpixels avec de bons résultats sans utiliser de post traitement corrigeant autant d'erreurs.

Les créateurs de SLIC ont placé leurs résultats dans un tableau pour pouvoir comparer plus facilement les différents algorithmes :

	Graph-based					Gradient-ascent-based				
	GS04 [8]	NC05 [23]	SL08 [21]	GCa10[b] [26]	GCb10[b] [26]	WS91 [28]	MS02 [4]	TP09[b] [15]	QS09 [25]	SLIC
Adherence to boundaries										
Under-segmentation error (rank)	0.23	0.22	-	0.22	0.22	-	-	0.24	0.20	**0.19**
Boundary recall (rank)	**0.84**	0.68	-	0.69	0.70	-	-	0.61	0.79	0.82
Segmentation speed										
320×240 image	1.08s[a]	178.15s	-	5.30s	4.12s	-	-	8.10s	4.66s	**0.36s**
2048×1536 image	90.95s[a]	N/A[c]	-	315s	235s	-	-	800s	181s	**14.94s**
Segmentation accuracy (using [11] on MSRC)	74.6%	75.9%	-	-	73.2%	-	-	62.0%	75.1%	**76.9%**
Control over amount of superpixels	No	Yes	Yes	Yes	Yes	No	No	Yes	No	Yes
Control over superpixel compactness	No	No	No	No[d]	No[d]	No	No	No	No	Yes
Supervoxel extension	No	No	No	Yes	Yes	Yes	No	No	No	Yes

FIGURE 6 – Table résumant les différents algorithmes de superpixels et leurs résultats

Voici également les tableaux des noms des algorithmes pour chaque abréviation donnée dans la table :

GS04	NC05	SL08	GCa10	GCb10
Felzenszwalb and Huttenlocher	The Normalized cuts algorithm	Moore et al	Veksler et al	

FIGURE 7 – tableau des noms des algorithmes de superpixels par méthode sur graphes

WS91	MS02	TP09	QS09
The watershed approach	Mean shift	The Turbopixel method	Quick shift

FIGURE 8 – tableau des noms des algorithmes de superpixels par méthode de gradient

On peut voir grâce à la table que SLIC est meilleur en tous points sur les métriques considérées par rapports aux autres (sauf de peu sur le boundary recall face à GS04 0.84 à 0.82). Il a des scores meilleurs mais assez proches par rapport aux tests d'Under Segmentation Error et de précision de Segmentation. SLIC se démarque des autres au niveau de la rapidité de segmentation où la différence de vitesse pour une même tâche entre SLIC et un autre algorithme est d'environ 10 à 50 fois plus rapide, que ce soit sur des images de petites ou grandes tailles. Enfin, SLIC est le seul de ces algorithmes à pouvoir à la fois contrôler le nombre et la compacité des superpixels, mais également étendre la segmentation en supervoxels(segmentation en trois dimensions).

En regardant ces résultats, il n'y a donc aucun doute sur le fait que SLIC est meilleur en tous points par rapports aux méthodes de superpixels de l'époque.

3 Principe de la recherche

Notre but était d'aller plus loin que l'algorithme SLIC actuel en utilisant la force du Deep Learning. Nous voulions rester sur le même fonctionnement que le SLIC classique, et pour pouvoir améliorer les résultats de segmentation, il faut trouver un moyen d'avoir de meilleurs superpixels, étant tous de formes relativement similaires et suivant au mieux les contours des images segmentées. La seule façon pour essayer d'améliorer ces points est de trouver un meilleur calcul pour D la distance permettant l'assignement du pixel à son centre le plus proche.

On sait que pour le SLIC classique, le pixel va appartenir au superpixel du centre le plus proche en fonction de sa distance mais aussi de sa couleur (si le pixel est rouge, et que deux centres sont à égale distance spatiale du pixel, mais que l'un est orange et l'autre bleu, le pixel va appartenir au groupe de pixels du centre orange). C'est ici qu'une piste peut être étudiée ; si à la place de la couleur, nous arrivons à trouver d'autres caractéristiques de similarité décrivant un pixel, alors ces caractéristiques pourraient servir pour le calcul de D et permettre un meilleur regroupement de pixels se ressemblant le plus possible, de façon plus précise qu'avec la couleur des pixels.

Nous voulons grâce aux réseaux profonds, pouvoir donner de nouvelles caractéristiques aux pixels d'une image, différentes des caractéristiques de couleur et de spatialité. Utiliser ce qui sort en dernière couche des réseaux ne sert à rien car nous voulons des matrices de la taille de l'image où chaque pixel possèdent des caractéristiques. Nous cherchons donc à obtenir les résultats d'un réseau en sortie de couche qui a effectué des calculs sur l'image à segmenter pour pouvoir l'utilise dans le calcul de D.

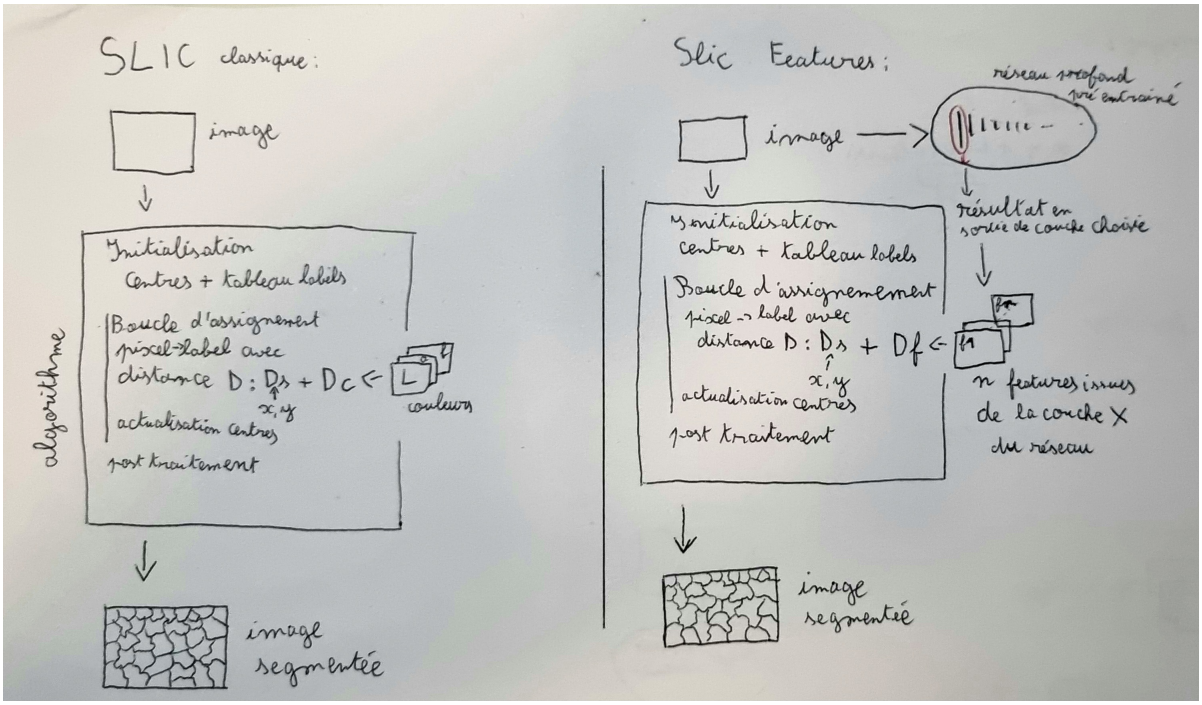

FIGURE 9 – Schéma simplifié de la différence SLIC classique et SLIC avec features

Ce schéma résume le principe de notre idée d'amélioration de performances de SLIC. Nous voulons faire passer en amont l'image que nous allons traiter dans un réseau de neurones, et extraire les données en sortie de la première couche, cela va nous donner plusieurs features (caractéristiques), qui permettent de caractériser l'image, où chaque feature est une matrice de la taille de l'image. Ces features vont permettre de modifier le calcul de D pour regrouper les pixels ayant des caractéristiques communes. Cette solution d'amélioration semble sur le principe prometteuse, nous avons donc développé cette idée pour confirmer si les résultats avec des features deep sont meilleurs que ceux de SLIC classique.

4 Extraction des features

Pour extraire des features il faut savoir ce qu'est un réseau. Pour notre cas, nous allons utiliser des réseaux de neurones convolutif. Il en existe différents et servent dans différents domaines.

4.1 Composition des réseaux convolutif

Les réseaux convolutifs (CNN) sont composés de plusieurs couches, comme tous les réseaux de neurones. Le but des CNN est principalement de segmenter, reconnaître et classifier des objets dans une ou plusieurs images. A l'entrée du réseau, une image passe par plusieurs couches successives de convolutions, avec différents filtres, puis de pooling qui va réduire la taille de l'image d'entrée tout en conservant les caractéristiques de celle-ci, jusqu'à ce que l'image devienne un vecteur. Après chaque couche de convolution, il y a une couche de correction ; elle permet de corriger certaines valeurs en sortie de convolution en appliquant une fonction qui va modifier les valeurs obtenues. La plus utilisé est la fonction reLU qui va remplaçer toutes les valeurs négatives par 0. Après plusieurs couches, quand l'image est devenu un vecteur, elle passe par une dernière couche la couche *Fully Connected*, elle va transformer le vecteur d'entrée en un nouveau vecteur, où chaque élément du vecteur sert de probabilité pour l'image de base, d'appartenir à telle ou telle classe. Les couches qui nous intéressent sont les couches de convolutions.

Une couche de convolution a pour but de faire passer un filtre sur un image pour en retirer des caractéristiques, sous formes de *feature maps*. Ce filtre est paramétrable pour permettre différentes convolutions. Pour balayer toute l'image, le filtre peut avoir un taille différente, le décalage (pas, stride en anglais) que va effectuer le filtre sur l'image est important, si le pas est égale à 1 alors les features en sortie seront de la même taille que l'image d'entrée. Enfin, des marges (padding) peuvent être ajoutées pour permettre au filtre de bien parcourir l'ensemble de l'image de façon homogène et aussi de permettre à contrôler la taille des features en sortie.

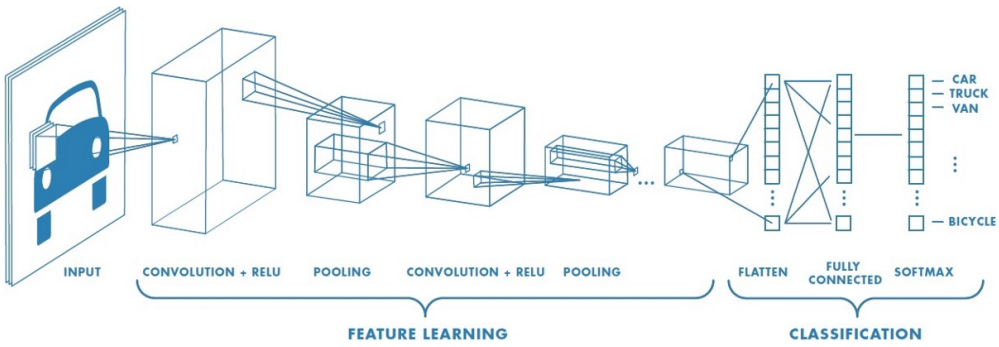

FIGURE 10 – Schéma simplifié d'un réseau de neurone convolutif

De façon abstraite, les features en sortie de convolution sont des filtres décrivant l'image en entrée, ce sont donc ces matrices que nous voulons utiliser pour SLIC. Il semble aussi important de ne pas trop descendre dans le réseau et de conserver les features en sortie des premières couches du réseaux car ce sont elles qui possèdent le plus de caractéristiques de bas niveau, utiles à décrire un image.

4.2 Le hooking

Généralement, un réseau est utilisé pour pouvoir utiliser les résultats qui sont donnés à la fin. Pour notre problème, ce qui se passe en fin de réseau ne nous intéresse pas, nous cherchons à prendre les données d'un réseau en sortie de couche. Pour cela nous utilisons un *hook*. En général, un hook est une fonction qui s'exécute automatiquement après un évènement. Un exemple simple de hook est quand l'on reçois une notification de notre banque sur notre téléphone lorsque nous avons effectué un achat sur internet.

Pour qu'un hook soit efficace avec un réseau, il faut que ce réseau soit pré-entraîné : un réseau pré-entraîné est un réseau avec ses paramètres fixes et qui s'est entraîné sur une large base de données pour effectuer des tâches (pour les CNN, la reconnaissance d'objets sur une base). Le réseau va donc déjà connaître les différents objets qu'il a vu lorsqu'il apprenait sur la base d'apprentissage. Dès lors, l'utilisation de ces réseaux pré-entraînés permet un apprentissage beaucoup plus rapide (il sera par exemple plus simple et rapide à un réseau d'apprendre à reconnaître un type d'animal si il sais déjà reconnaître plusieurs animaux, que de partir de 0).

Un hook sur un réseau pré-entraîné va alors permettre l'extraction de features sans avoir besoin de modifier le réseau, et va permettre de s'arrêter quand nous avons extrait ce dont nous avons besoin. Les différents frameworks de machine learning (Pytorch, Tensorflow) ont créés des modules de hook qui peuvent être utilisés facilement, en appelant une méthode du module et en indiquant le réseau et la couche souhaité. En utilisant la méthode paramétré sur une image, nous obtenons les features extraites en fin de couche souhaité.

Nous pouvons enfin utiliser les données récoltées et les manipuler, sans avoir besoin de l'utilisation d'un réseau ultérieurement. Cette technique permet d'éviter

le temps que mets un réseau à s'entraîner sur une image d'entrée car nous nous arrêtons lors des premières couches, car ce sont elles les plus intéressantes.

5 Implémentation

Pour pouvoir obtenir nos résultats et valider notre hypothèse, nous avons implémenté notre propre SLIC, qui est classique, puis nous avons voulu implémenté l'extracteur des features d'un réseau. Nous avons ensuite ajuster notre implémentation de SLIC pour qu'il puisse utiliser nos features. Pour cela seulement deux fichiers ont été créés, un pour SLIC et l'autre pour l'extraction, tous les deux codés en langage Python.

5.1 implémentation de SLIC

Pour implémenter SLIC, nous nous sommes référé à l'algorithme de base de SLIC à la figure 1. Nous avons défini une classe *Cluster* qui a pour attribut ses coordonnées et sa couleur Lab. Il possède également un numéro qui est son index. A chaque création d'un cluster, son numéro est incrémenté.

Une seconde classe est créée, la classe SLICProcessor, qui va contenir toutes les méthodes de pour segmenter une image en superpixels. Cette classe a pour attribut K, le nombre de superpixels souhaité, et m l'indice de compacité. Plusieurs mesures sont initialisées ici, comme l'image, ses dimensions, son nombre de pixel et aussi $S = \sqrt{N/k}$. Nous initialisons deux matrices de la taille de l'image, une remplie ∞ et l'autre de -1, ce sont d et l. Pour initialiser les centres de chaque cluster, la méthode *init_clusters()* va créer K objets de la classe Cluster avec seulement leurs coordonnées x, y, avec un pas de S.

Pour bouger les centres des clusters au points ou les gradients sont les plus bas, la méthode *move_clusters()* parcours la liste des clusters et change les coordonnées x, y en fonction des coordonnées où le gradient est le plus petit. Ce calcul de gradient est fait dans une autre méthode et est appelée dans *move_clusters()*. En actualisant les coordonnées des objets clusters, c'est ici aussi que nous assignons leurs valeurs de couleurs.

Nous passons maintenant à la boucle principale de l'algorithme ; notre méthode *assignment_and_update_cluster_c()* va boucler dans notre liste de cluster. pour chaque centre de cluster, nous prenons les valeurs L,A,B de chaque pixel voisin de rayon $2S$. Ils serviront aux calcul de $Dc = (L - C_l)^2 + (A - C_a)^2 + (B - C_b)^2$ et $Ds = (x - C_x)^2 + (y - C_y)^2$. Nous additionnons les deux distances avec $D = Dc + Ds * ms$ avec $ms = \frac{m^2}{S^2}$. Enfin, si $D < d_{x,y}$ alors $d_{x,y}$ prend la valeur D et $l_{x,y}$ prend la valeur cluster_label, qui est une valeur qui s'incrémente quand on passe au cluster suivant dans la liste des clusters. le tableau l (que nous appelons ltab) possède pour chaque pixel la valeur de son label. Ce tableau est la carte des labels qui servira pour nos calculs prouvant ou non l'efficacité de segmentation.

Pour actualiser les centres des clusters, nous initialisons des listes de longueur K, pour x,y,l,a,b. En parcourant une copie de ltab, on incrémente la valeur x,y,l,a ou

b à l'index correspondant à la valeur du label de ltab dans chaque liste. En divisant chaque valeur obtenu par le nombre de pixels appartenant à un label, nous obtenons la nouvelle valeur moyenne de x,y,l,a,b pour chaque cluster. Nous actualisons notre liste de clusters avec ces nouvelles valeurs pour actualiser les centres.

L'assignation et l'actualisation sont imbriquées dans un boucle allant de 0 à 3 ; en effet nous répétons 4 fois les deux actions pour que les centres et les clusters convergent dans la meilleur position. L'algorithme original utilise un seuil et un calcul d'erreur résiduelle, mais en faisant plusieurs essais, nous nous sommes rendu compte que entre 3 à 4 boucles sont nécessaires pour arriver à convergence, avec K plus ou moins grand.

Une méthode de sauvegarde d'image est créée où les valeurs des pixels sont attribué en fonction de la valeur du centre du cluster dans lequel il est.

Nous avons créé la méthode *create_connectivity()* qui va permettre de corriger les erreurs de segmentation. Une sauvegarde de ltab est faite car elle montre le résultat de la segmentation avant post traitement. Le principe de cette méthode est de parcourir une copie de ltab sous forme de liste, de garder en mémoire les voisins des labels autour d'un pixel. Si les pixels sont peu nombreux alors ils prennent comme valeur l'index de leurs voisins. A la fin, ltab prend les valeurs de la liste qui a été modifié. Cette méthode modifiant le nombre de clusters, il faut modifier K et actualiser la taille de la liste des clusters. Il faut donc repasser pas une méthode d'actualisation des valeurs des clusters avec un fonctionnement similaire que pour la phase d'actualisation dans la méthode *assignment_and_update_cluster_c()*. Une sauvegarde du nouveau ltab est faite qui est le résultat de la segmentation après post traitement.

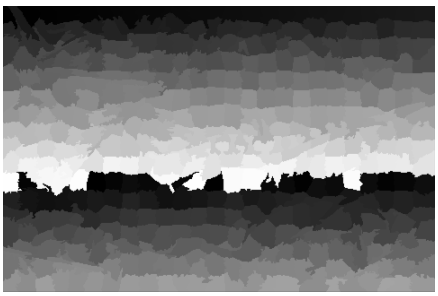

FIGURE 11 – Exemple de tableau ltab, chaque teinte représente un label

Une méthode appelée *iterate()* appele les méthodes d'initialisation des clusters, leur déplacement et d'assignement et actualisation des clusters. C'est ici que les images visuelles sont sauvegardées, une avant et une après lancement de la méthode de post traitement. Enfin dans le *"main"* du fichier, nous initialisons m à la valeur souhaitée, puis créons un objet *SLICProcessor* avec comme paramètre d'entrée, l'image à segmenter, K, et m. Enfin nous faisons appel à la méthode *iterate()* qui va lancer les autres méthode qui vont segmenter l'image choisie et sauvegarder ses résultats.

5.2 implémentation de l'extraction des features

Avec notre SLIC opérationnel, nous voulions pouvoir extraire des features issues de réseaux profond. Nous avons utilisé Pytorch, le framework de machine learning, développé par Meta. Pytorch possède plusieurs modèles de réseaux pré-entrainés. Nous avons choisi de prendre 3 réseaux pour montrer les performances du Deep Learning.

- **VGG** : Développé en 2014, il a marqué une progression par rapport aux modèles de réseaux précédents en proposant, dans les couches de convolution, des noyaux de convolution de plus petites dimensions.
- **ResNet** : Développé en 2015, le réseau de neurones résiduel, est beaucoup plus profond que les anciens modèles de réseaux, il est caractérisé par des *Skip connections*, le fait de sauter certaines couches du réseau et passer à celles qui suivent.
- **EfficientNet** : CNN qui a la particularité de se mettre à l'échelle de dimension ou de profondeur, grâce à un *compound coefficient*, en fonction du problème à résoudre.

Ces trois réseaux sont très populaires et possèdent de très bons scores pour la classification d'image sur de grandes bases comme ImageNet. Grâce aux outils de Pytorch, extraire les features d'un réseau était simple.

Nous definissons une classe appelée FeatureExtractor avec comme attributs, le nom d'un modèle de réseau, et le nom de la couche où nous voulons extraire les features. Pour la couche choisie, une méthode Pytorch va faire un hook, ce qui va permettre d'attraper ce qu'il sort à la couche voulue, et les sauvegarder. Nous créons donc un objet FeatureExtractor avec comme paramètre le nom du réseau que nous voulons et le nom de sa couche. Voulant les sorties des premières couches de convolution des réseaux, nous prenions comme nom les couches que nous souhaitions. Nous avons ensuite a appliquer cet objet à une image pour obtenir différentes features de notre image.

En faisant des essais, nous nous rendons compte que les features obtenues ne sont pas de bonnes tailles, en fonction du réseau et de la couche utilisée. C'est logique, chaque réseau a une couche de convolution différente, avec un noyau différent. Nous avons regardé les paramètres de la couche souhaitée des différents réseaux et les avons placées dans un tableau.

	Nombre de features en sortie	taille du noyau	stride	padding
VGG	64	3	1	1
EfficientNet	32	3	2	1
ResNet	64	7	2	3

FIGURE 12 – Paramètres de la première couche de convolution des différents réseaux pré-entraînés utilisés

Nous avons choisi d'extraire les données sorties des premières couches de convolution des réseaux, nous aurons des features de bas niveau, qui peuvent caractériser

une image. Nous voulions également que les features en sorties soient de la taille de l'image d'entrée, pour cela, nous avons fait un pré-traitement de l'image d'entrée avant de servir pour obtenir nos features. Des méthodes prévues par Pytorch ont été utilisée ; après avoir convertie l'image en tenseur pour être utilisable par les outils Pytorch, nous appliquons un padding (ajout de pixels au bords de l'image) supplémentaire de taille 1 à l'image, cela permet d'amoindrir les effets de bords(pixels aux bords de l'image avec de mauvaises valeurs).

Nous redimensionnons ensuite la taille de l'image en la multipliant par le stride (pas du noyau lors du balayage de l'image) du réseau, en effet si un réseau a un padding de 2, les features en sorties seront 2 fois plus petite, un redimmensionnement est donc nécessaire pour avoir des tailles similaires entre l'image d'entrée sans pré-traitement et les features en sortie. la méthode *tab_features()* permet d'englober le pré-traitement, et la récupération des features. Cette méthode renvoie les features de la couche du réseau choisi.

FIGURE 13 – Image originale

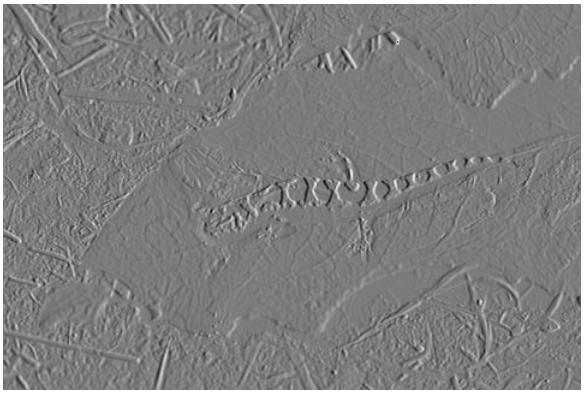

FIGURE 14 – Une feature issue du réseau Efficientnet de l'image originale

5.3 ajout des features à SLIC

Pour associer les 2 parties, il a fallu modifier les 2 fichiers de façon à pouvoir créer un SLIC opérant sur des features deep. Nous nous sommes rendu compte qu'il était impératif de normaliser les valeurs des features pour le calcul de D de SLIC, il faut pouvoir faire des calculs entre des valeurs de la même unité. Dans le fichier d'extraction, nous avons normalisé les valeurs des features en les ramenant entre 0 et 1 et en arrondissant les valeurs proches de 0 à 0. Nous nous sommes par la suite rendu compte que des features sorties étaient nulles, nous les avons donc supprimées de l'ensemble des features pour ne pas fausser le calcul de D.

le reste se déroule dans le fichier pour SLIC. Vu que les clusters sont maintenant définis aussi par des features, nous avons initialisé des features nulles dans la composition de la classe *Cluster*. la méthode *tab_features()* pour obtenir les features d'un réseau souhaité est appelée dans la classe *SLICProcessor*. Dès que

l'utilisation de la liste des clusters intervient dans le fichier, pour les actualisations par exemple, il a fallu rajouter les features *f*. Pour pouvoir choisir entre la segmentation de SLIC classique et celle des features, nous avons créé la méthode *assignment_and_update_cluster_f()* qui est très similaire à *assignment_and_update_cluster_c()*. Sa différence réside sur le calcul de D. Ici, $D = Df + Ds * ms$, Ds reste le même que pour SLIC couleurs, et $Df = cf * \sum(f - C_f)^2$, avec f les valeurs des features du pixel et C_f les valeurs des features du centre du cluster. Nous avons aussi $cf = \frac{2*255^2+100^2}{\sum(\min f - \max f)^2}$. cf sert de coefficient de normalisation, les valeurs des features sont de même unité que les valeurs L,a,b. Cette normalisation permet donc au calcul juste de D et permet donc de pouvoir comparer les différents SLIC lors de notre analyse. Avec cette méthode, nous pouvons enfin tester une segmentation en superpixels avec des features deep.

Pour pouvoir choisir entre SLIC couleurs et SLIC features, un paramètre dans la méthode *iterate()* a été ajouté. Enfin, nous avons ajouter un parseur pour pouvoir mettre différents paramètres en ligne de commande pour lancer le fichier et choisir les différents paramètres de lancement, et pouvoir servir à une automatisation lors de la création de résultats pour l'analyse de nos recherches. Les paramètres sont, le nom de l'image, le nombre de superpixels, SLIC couleurs ou Features, le nom du réseau et la couche. Les SLIC étant opérationnel, nous pouvons enfin faire des expérimentation pour voir si l'utilisation de features deep issues de réseaux va rendre de meilleurs résultats pour SLIC.

6 Expérimentation

Après avoir fini la conception de notre SLIC avec Deep Features, il fallait pouvoir le tester sur une base d'essais, et ce avec différentes images mais également différents calculs de qualité de segmentation. Notre but est donc de savoir si les features apportent une meilleure segmentation par rapport au SLIC classique.

6.1 Benchmarks

Nous avons utilisé la même base d'essais que celle des travaux de Achanta *et al.*, à savoir la BSDS300. Elle est appelée *Berkeley Segmentation Dataset* et possède 300 images en couleur de test, toutes de même taille, et choisies spécialement pour tester si des algorithmes de segmentation fonctionnent bien sur différentes formes et textures. Pour chaque image, une carte des contours est aussi présente, ces cartes sont l'addition de segmentations effectuées par plusieurs personnes qui ont dessiné à la main les contours d'une image. Cette base a été conçue par l'université de Berkeley et elle est utilisée pour faire des recherches en segmentation d'images et détection de contours. Cette base est idéale pour pouvoir faire nos essais.

6.2 Protocole expérimental

Nous voulons tester les performances de SLIC classique face à SLIC Features, nous allons utiliser 5 différents calculs de performance de segmentation :

- **Le Boundary Recall** : il vérifie les différences de contours par rapport à une vérité terrain. Plus il est proche de 1 meilleur est le résultat, ce qui veut dire que les superpixels suivraient parfaitement les contours de l'image. Sa formule est $R = \frac{TP}{TP+FN}$ avec TP le nombre de pixels des contours de la vérité terrain qui est proche d'un pixel des contours de l'image segmentée, et FN le nombre de pixels des contours de la vérité terrain qui n'est pas proche d'un pixel des contours de l'image segmentée.
- **L'UnderSegmentation Error(USE)** : Il détecte selon une vérité terrain si des pixels sont regroupés ensembles alors qu'ils ne devraient pas. L'erreur doit donc tendre vers 0 pour avoir un bon résultat. Sa formule est $\frac{1}{N}\sum_{G_i \in G}\sum_{S_j \cap G_i \neq 0} \min |S_j \cap G_i|, |S_j - G_i|$, avec N le nombre de pixels, S l'image segmentée et G la vérité terrain.
- **L'Achievable Segmentation Accuracy(ASA)** : Il permet de détecter selon une vérité terrain si les pixels labellisé appartienent aux bons superpixels. Il doit tendre vers 1. Sa formule est $\frac{1}{N}\sum_{S_j \in S} \max_{G_i} |S_j \cap G_i|$
- **La Contour Density et Compactness(CDC)** : Permet de mesurer la densité et la compacité des superpixels.
- **La Mean Squared Error(MSE)** : Correspondant à l'erreur quadratique moyenne, elle correspond ici à la différence entre l'image créée et l'image estimée. Minimiser la MSE revient à se rapprocher de l'image d'origine.

Avec ces calculs, il nous fallait pouvoir créer des graphiques pour chaque SLIC et chaque calcul différent, pour pouvoir comparer facilement les performances des SLIC. Nous avons créé plusieurs fichiers de script pour pouvoir automatiser le lancement des SLIC avec plusieurs paramètres, puis créer les graphiques, voici dans quel ordre ont été lancé les scripts ;

1. Un script de calcul des différents SLIC avec différents paramètres. Ce script va lancer SLIC classique, SLIC VGG, SLIC resnet et SLIC efficientnet, sur les 300 images de la BSDS300, avec K le nombre de superpixels souhaité, variant de 50 à 600 avec un pas de 50. Nous avons fixé m l'indice de compacité à 40 car il permet aux superpixels de rester compact et de suivre les contours. Nous obtenons donc les résultats des segmentations, avant et après post traitement sous forme d'image .png, et aussi les cartes des labels associés, avant et après post traitement également, sous format .cimg.
2. Un script va lire les fichiers de verité terrain SEG contenant les contours des images de la BSDS300 et les recopier sur des fichiers .cimg, ceci servira pour pouvoir exécuter nos fichiers de calculs qui prennent en entrée des fichier CIMG.
3. Un script qui va effectuer tous les calculs pour chaque image de chaque SLIC avec un K donné. Il crée plusieurs fichiers CSV (tableur) en sortie qui auront le nom du SLIC utilisé, le calcul utilisé, le K donné, et donc 300 lignes contenant une valeur à chaque ligne. Ce script effectue une seconde action. Avec tous les fichiers .csv , les 300 valeurs d'un fichier est moyenné en 1 seule qui va être stockée dans un fichier ayant pour nom le calcul, le SLIC utilisé, avant ou après post traitement. Avec les différents K, nous avons donc un fichier .csv contenant pour une ligne, K le nombre de superpixel et son score moyen.

4. Un script qui utilise les fichiers avec les valeurs moyennées du script précédent et va créer pour chaque calcul, un graphe contenant en abscisse K le nombre de superpixel, et en ordonnée le score du calcul. Sur ce graphique, il y a les courbes pour chaque SLIC, une courbe est créée grâce aux données issue d'un fichier.
5. Un script qui va également créer les mêmes courbes que le script précédent, mais va sortir deux graphiques, un pour les résultats des SLIC avant post traitement, et le deuxième pour les résultats après post traitement.

Nous nous retrouvons avec chaque image de la BSDS300 segmentée en superpixels par SLIC classique (couleurs), SLIC avec des features deep issu du réseau VGG, SLIC resnet et SLIC efficientnet, avec un nombre de superpixels allant de 50 à 600 avec un pas de 50. Nous avons également les cartes des labels de chaque image correspondante. Nous avons également des fichiers tableurs de chaque résultat de calcul pour chaque image et enfin, pour chaque calcul, des graphiques avant/après post traitement, de chaque SLIC. Il nous reste donc à analyser les résultats pour dégager une conclusion à nos recherches.

6.3 Résultats et analyse

Grâce à nos différents scripts, nous avons des résultats visuels avec les images segmentées pour chaque SLIC, mais aussi des graphiques pour voir les différents scores des calculs de chaque SLIC.

Intéressons nous d'abord aux résultats des images segmentées, où nous pouvons seulement supposer de meilleurs ou moins bon résultats.

FIGURE 15 – Image de base

FIGURE 16 – SLIC resnet K=100

FIGURE 17 – SLIC resnet K=300

FIGURE 18 – SLIC resnet K=500

En regardant K le nombre de superpixel avec la figure 15, nous voyons que plus K est grand, meilleur semble être l'image segmentée. Cela semble logique vu que avec plus de clusters, nous pouvons obtenir plus de détails. Nous avons pris un exemple au hasard, ici avec SLIC resnet mais pour les autres SLIC, sur d'autres images, K semble agir de la même manière, ce qui est encourageant.

FIGURE 19 – Image de base

FIGURE 20 – SLIC couleurs K=500

FIGURE 21 – SLIC resnet K=500

FIGURE 22 – SLIC efficientnet K=500

FIGURE 23 – SLIC vgg K=500

Pour la figure 19, nous regardons les résultats de segmentation des 4 SLIC avec le même K. Il devient plus compliqué de savoir laquelle est la meilleur segmentation, on peut cependant noter que pour SLIC couleur et SLIC VGG, les rayures n'ont pas réussi à être conservées, au niveau du corps du tigre pour cet exemple. En ayant regardé d'autres images, on a pu voir que les images de SLIC couleurs conservent moins bien certains contours car on s'aperçoit que ses clusters se déforment moins que les clusters des SLIC à features deep, mais donc restent plus compactes.

Pour avoir des preuves irréfutables des performances de nos SLIC, il faut regarder les courbes de nos calculs et voir quel SLIC est le plus performant.

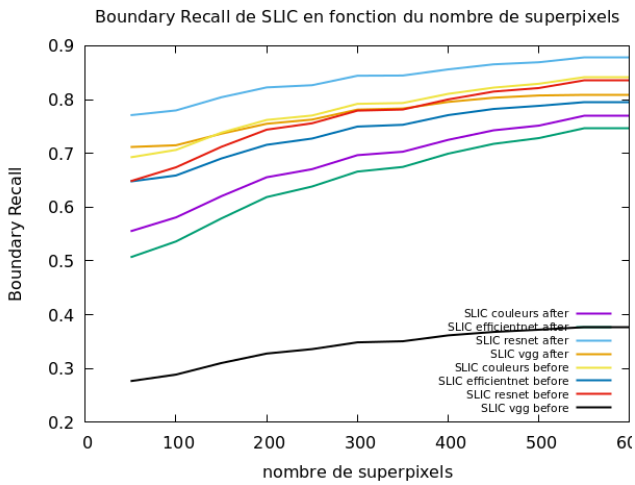

FIGURE 24 – Boundary Recall des SLIC

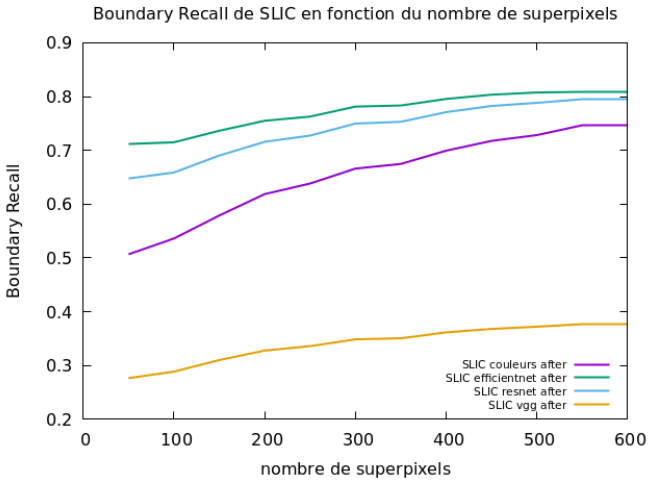

FIGURE 25 – Boundary Recall des SLIC après post traitement

La première mesure est le Boundary Recall à la figure 24, en voulant interpréter ce graphique, nous remarquons que tous les SLIC après post traitement ont un meilleur score que leur équivalent avant post traitement. En ayant vérifié les autres graphiques, la tendance est la même donc cela prouve que le post traitement est efficace et indispensable à la qualité des résultats. Nous allons donc pour les autres graphiques seulement montrer les résultats après post traitement pour plus de clarté comme à la figure 25.

Ici, nous savons que plus le Boundary Recall s'approche de 1, plus la segmentation suit les contour de l'image originale. Plus K augmente, plus le Boundary Recall augmente ce qui semble normal, nous avons SLIC resnet et SLIC efficientnet qui ont des résultats assez proches, s'approchant d'un score de 0.8 avec K=600, avec l'efficientnet meilleur, vient ensuite SLIC couleur qui à un score au mieux de 0.7 et enfin SLIC VGG qui ne dépasse pas un score de 0.4. On peut voir que SLIC resnet et efficientnet sont donc pour le moment plus performant que le SLIC classique pour suivre les contours d'une image, ce qui est encourageant. Pour SLIC VGG avec un score aussi mauvais, le problème doit être que les features extraites pour VGG n'ont pas été bonnes.

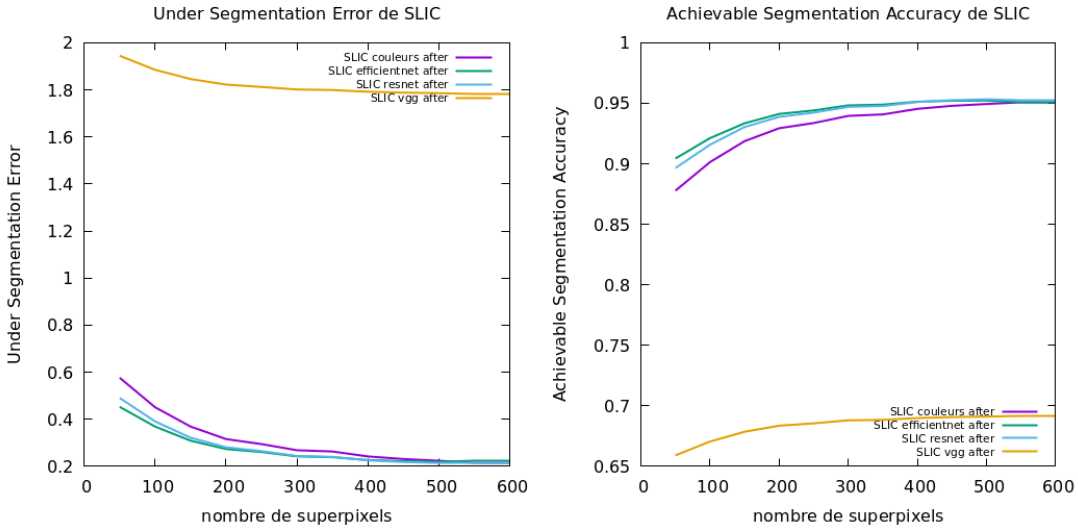

FIGURE 26 – Under segmentation error and Achievable segmentation accuracy après post traitement

En oubliant encore les résultats aberrant de SLIC VGG à la figure 26, les scores des autres SLIC sont interprétable. Comme pour le Boundary Recall, Nous avons SLIC efficientnet et SLIC resnet qui sont meilleurs que SLIC couleurs, avec SLIC efficientnet un peu devant. On peut voir qu'en augmentant K, l'erreur de segmentation diminue jusqu'à 0.2 pour les 3 SLIC et pour la précision de segmentation, elle augmente jusqu'à 0.95 pour les 3. SLIC classique semble toujours un peu moins performant que les SLIC avec deep features.

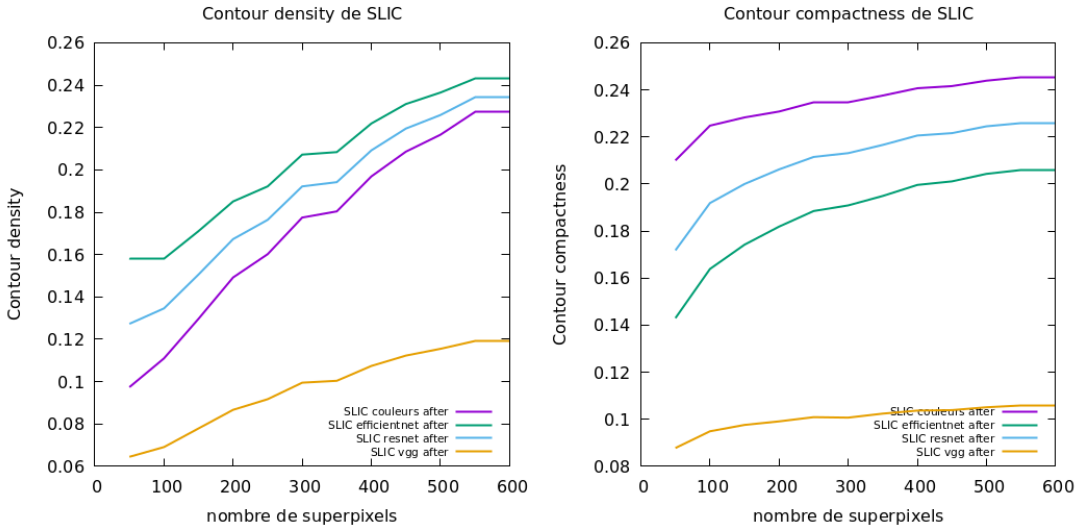

FIGURE 27 – Contour density and compactness après post traitement

Pour la figure 27, avec encore de très mauvais résultats pour SLIC VGG, on ne peut malheureusement plus tirer de conclusion à son sujet, une erreur lors de la segmentation ou des calculs spécifiquement pour SLIC VGG, à du apparaître et fausser les résultats. Pour ce qui est de la densité, nous retrouvons les mêmes résultats qu'auparavant, SLIC efficientnet à un indice de densité plus grand que celui de SLIC resnet qui lui même est plus grand que SLIC classique. Cependant pour la compacité, l'ordre est inversé et SLIC couleurs à un meilleur indice.

Nous l'avions un peu remarqué de façon visuelle, les superpixels de SLIC couleurs semblaient plus compactes et moins déformés que les autres, nous en avons désormais la preuve. Ces résultats semble corréler avec les autres, vu que les clusters de SLIC classiques se déforment moins, alors ils peuvent moins s'adapter aux contours comme le font les SLIC deep features, où ils sont meilleurs.

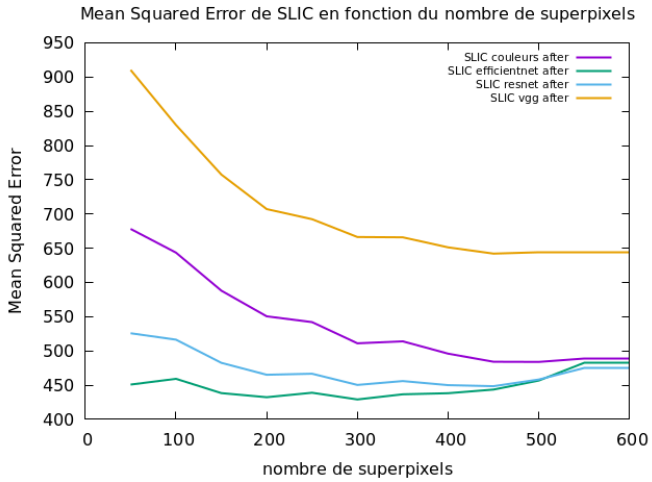

FIGURE 28 – Mean Squared Error après post traitement

La figure 28 nous montre la MSE, avec un petit K, elle est très elevé pour le SLIC classique et est de 680, elle baisse par la suite en augmentant K pour atteindre 500. La MSE de SLIC resnet débute vers 530 pour baisser à 460 lorsque K= 500 mais remonte ensuite pour rejoindre SLIC classique. SLIC efficientnet commence le plus bas en restant autour de 450 jusqu'à K=500 pour ensuite également remonter vers 500 comme les deux autres. Vu que la MSE est à minimiser, SLIC efficientnet est encore une fois meilleurs que les autres.

Grâce aux différents graphiques, nous pouvons affirmer que les SLIC Deep Features offrent de meilleurs résultats, non négligeables, que SLIC classique. Ils font moins d'erreurs de segmentation et permettent de mieux suivre les contours, en ayant pour seul bémol d'avoir des superpixels moins compacts, ce qui reste intéressant pour la segmentation. Utiliser des outils de Deep Learning permet donc d'améliorer les résultats de SLIC. Notre idée d'utiliser des features dans le calcul de distance etait bonne, c'est efficace et cela permet bien une meilleur segmentation. Nous avons vu que efficientnet est également un peu meilleur que resnet pour notre cas.

Un meilleur réseau est peut être disponible pour pouvoir sortir des features qualitatives et avoir des résultats satisfaisants, il en va de même pour la couche utilisée, ici nous avons utilisé la première couche de convolution de nos réseau, en utilisant une couche plus profonde nous aurions eu des features de plus haut niveau. Les petites différences de résultats entre efficientnet et resnet doivent venir de la formation de leur couche de convolution, la première couche d'efficientnet à un plus petit noyau et un plus petit padding que celui de resnet, si nous avions les paramètres optimales pour une couche, nous aurions les meilleures features possibles à utiliser pour SLIC et obtenir de meilleurs résultats.

Conclusion

L'utilisation du Deep Learning étant en vogue, nous voulions voir si son utilisation permettait l'amélioration d'algorithmes n'en utilisant pas. SLIC était un candidat pour pouvoir faire des essais. En utilisant des features deep issues de réseaux performant, nous avons modifié SLIC pour pouvoir segmenter en prenant en compte ces nouvelles données. A la suite d'expérimentations et de calculs montrant l'efficacité de segmentation, les résultats issus des SLIC features étaient meilleurs que ceux de SLIC classique. L'idée de vouloir utiliser la puissance du Deep Learning pour permettre une amélioration de performances était bonne.

De nombreux travaux mélangeant segmentation et Deep Learning ont lieux, c'est le cas du *SSN*(Superpixel Sampling Network, de Varun Jampani *et al.*), où une image passe dans un réseau de neurones profond spécifique pour extraire des caractéristiques et ensuite être utiliser avec un SLIC également modifié. Avec des réseaux de neurones encore plus efficaces et conçus uniquement pour certaines tâches précises, nous pourrions optimiser encore plus certains outils informatiques et progresser dans des domaines utiles à tous, comme par exemple dans l'analyse de données et la détection de maladies précoces pour la santé.

A titre personnel, j'ai pu découvrir le travail dans un laboratoire de recherche, ce qui est différent d'une entreprise privée, cela m'a permis de devenir plus autonome pour travailler sur un projet et montrer un résultat, qu'il soit concluant ou non. J'ai apprécié développer ce projet car le thème du Deep Learning est d'actualité, et je suis satisfait car j'ai pu donner un résultat sur une question qui a été posé par des chercheurs, ce résultat peut donc être utile pour mener à bien d'autres recherches. Enfin, en ayant oeuvré tout au long du stage, j'ai pu améliorer mes connaissances et ma pratique sur le Deep Learning et ses outils, ce qui ne peut qu'être bénéfique pour mener à bien d'autres projets axés sur ce sujet.

Références

[1] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, Sabine Süsstrunk: SLIC Superpixels Compared to State-of-the-Art Superpixel Methods, 2012

[2] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, Sabine Süsstrunk: SLIC Superpixels, 2010

[3] Latinxao: slic-python-implementation

[4] RaelagF: SLIC_superpixel_segmentation

[5] Radhakrishna Achanta: SLIC

[6] Pytorch

[7] Frank Odom: Pytorch Hooks

[8] Torchvision models

[9] Convolutional Neural Networks: Padding and Stride

[10] David Stutz: Superpixel Benchmark

[11] The Berkeley Segmentation Dataset and Benchmark (300)

[12] Raghav Prabhu: understanding of convolutional neural network