



Projet de Fin de Module : La Vision par Ordinateur

**Application des Modèles de Segmentation
pour la Détection des Tumeurs Cérébrales :
Analyse Comparative par Apprentissage en
Profondeur**

Réalisé par :

BADI Oumaima

BENLAGOTE Zainab

ELYAZIJI Boutaina

Filière :
M2SI

Encadré par :

Mr. OUSGUINE Said

Soutenu le .. Décembre 2023

Remerciements

Nous tenons également à exprimer notre gratitude envers notre équipe pour sa collaboration exceptionnelle et ses efforts collectifs tout au long de la réalisation de ce projet. Chaque membre a apporté des contributions précieuses, travaillant avec détermination et partageant des idées novatrices qui ont été essentielles à la réussite de ce projet.

C'est grâce à l'orientation bienveillante du Professeur Ousguine et à l'engagement de notre équipe dans un esprit collaboratif que nous avons pu aborder ce projet avec succès. Nous tenons à souligner l'importance de cette expérience collaborative qui a enrichi notre parcours académique. *Nous souhaitons adresser nos sincères remerciements au Professeur Saïd Ousguine pour son engagement constant et son dévouement tout au long du semestre dans le cours de Vision par Ordinateur. Les connaissances partagées et la méthodologie pédagogique du Professeur Ousguine ont été d'une importance cruciale pour notre compréhension approfondie des concepts.*

Nous tenons également à exprimer notre gratitude envers notre équipe pour sa collaboration exceptionnelle et ses efforts collectifs tout au long de la réalisation de ce projet. Chaque membre a apporté des contributions précieuses, travaillant avec détermination et partageant des idées novatrices qui ont été essentielles à la réussite de ce projet.

C'est grâce à l'orientation bienveillante du Professeur Ousguine et à l'engagement de notre équipe dans un esprit collaboratif que nous avons pu aborder ce projet avec succès. Nous tenons à souligner l'importance de cette expérience collaborative qui a enrichi notre parcours académique.

Table des matières

1	<i>Introduction</i>	8
1.1	<i>Contexte du Projet</i>	8
1.2	<i>Problématique</i>	8
1.3	<i>Objectif de l'Étude</i>	8
1.4	<i>Structure du Rapport</i>	9
2	<i>Segmentation des IRM cérébraux dans la littérature</i>	10
2.1	<i>Introduction</i>	10
2.1.1	<i>Définition de la segmentation</i>	10
2.1.2	<i>Les objectifs de la segmentation</i>	11
2.2	<i>État de l'art du point de vue « Traitement d'images »</i>	12
2.2.1	<i>Segmentation basée sur les régions</i>	12
2.2.1.1	<i>Méthodes de croissance de régions</i>	12
2.2.1.2	<i>Méthode de Watershed</i>	13
2.2.1.3	<i>Méthode de Région-Merging</i>	15
2.2.2	<i>Segmentation basée sur les contours</i>	16
2.2.2.1	<i>Méthodes dérivatives</i>	16
2.2.2.2	<i>Modèles déformables</i>	16
2.2.3	<i>Limites et défis rencontrés avec ces méthodes</i>	17
2.2.3.1	<i>Méthodes de régions</i>	17
2.2.3.2	<i>Méthodes de contours</i>	17
2.3	<i>État de l'art du point de vue « Reconnaissance des formes »</i>	18
2.3.1	<i>Transition vers l'Apprentissage en Profondeur</i>	19
2.3.1.1	<i>Définition de l'Apprentissage en Profondeur</i>	19

2.3.1.2 <i>Utilisation de l'Apprentissage Profond dans la Segmentation</i>	19
2.3.2 <i>Techniques Actuelles de Segmentation avec les Réseaux de Neurones</i>	19
2.3.3 <i>Avantages et Limitations</i>	20
2.3.4 <i>Conclusion</i>	20
3 <i>Approches de Segmentation pour la Détection des Tumeurs Cérébrales</i>	21
3.1 <i>Introduction</i>	21
3.1.1 <i>Segmentation d'image et segmentation sémantique</i>	21
3.1.2 <i>Historique de la segmentation sémantique</i>	22
3.1.3 <i>Réseaux de Neurones Convolutifs Profonds dans la Segmentation</i>	23
3.1.3.1 <i>Limitations des DCNN dans la Segmentation</i>	23
3.1.4 <i>Choix des Algorithmes pour Notre Étude</i>	25
3.2 <i>Segmentation à l'aide de l'Architecture DeeplabV3+</i>	25
3.2.1 <i>Évolution de Deeplab</i>	25
3.2.2 <i>Convolution Atrous</i>	25
3.2.3 <i>Deeplabv1</i>	26
3.2.4 <i>Deeplabv2</i>	27
3.2.5 <i>Deeplabv3</i>	28
3.2.6 <i>Deeplabv3+</i>	28
3.3 <i>Segmentation à l'aide de l'Architecture PSPNet</i>	29
3.3.1 <i>Architecture Encodeur-Décodeur pour la Segmentation Sémantique</i>	30
3.3.1.1 <i>Encodeur PSPNet</i>	30
3.3.1.2 <i>Décodeur PSPNet</i>	32
3.4 <i>Segmentation à l'aide de l'Architecture SegNet</i>	33
3.5 <i>Segmentation à l'aide de l'Architecture Unet</i>	35
3.5.1 <i>Le chemin de contraction ou de sous-échantillonnage (Encodeur)</i> :	36
3.5.2 <i>Le goulot d'étranglement horizontal</i> :	36
3.5.3 <i>Le chemin d'expansion ou de suréchantillonnage (Décodeur)</i> :	36
3.6 <i>Segmentation à l'aide de l'Architecture Unet++</i>	37
3.6.1 <i>Couches de Convolution sur les Chemins de Saut</i> :	37
3.6.2 <i>Connexions de Saut Denses</i> :	37
3.6.3 <i>Supervision Profonde</i> :	38
3.6.3.1 <i>Chemins de Saut Redessinés</i>	38
3.6.3.2 <i>Supervision Profonde</i>	38

3.7 Segmentation à l'aide de l'Architecture Attention-Unet	38
4 Évaluation Comparative des Modèles de Segmentation	41
4.1 Présentation des outils utilisés	41
4.1.1 Environnement d'exécution	41
4.1.2 Bibliothèques et frameworks	41
4.1.3 Langage de programmation	42
4.2 Base de données	42
4.2.1 Description détaillée de la base de données	42
4.2.2 Composition de l'échantillon	43
4.2.3 Annotations et divisions	44
4.2.4 Caractéristiques génomiques	44
4.2.5 Prétraitement des données	45
4.3 Les métrics de performance	45
4.3.1 Indice de Similarité Jaccard (IOU)	46
4.3.2 Coefficient Dice (Dice Coef)	47
4.3.3 Précision (Precision)	48
4.3.4 Rappel (Recall)	49
4.4 La fonction du coût	49
4.4.1 Dice Loss	49
4.4.2 L'objectif	50
4.5 Les optimiseurs	50
4.6 Implémentation des Méthodes	51
4.6.1 DeepLabV3+	51
4.6.1.1 L'entraînement du modèle	54
4.6.1.2 Les résultats de l'entraînement	55
4.6.1.3 L'évaluation du modèle	58
4.6.1.4 Tester le modèle sur les images du test	58
4.6.2 PSPNet	58
4.6.2.1 L'entraînement du modèle	61
4.6.2.2 Les résultats de l'entraînement	62
4.6.2.3 L'évaluation du modèle	63
4.6.2.4 Tester le modèle sur les images du test	63
4.6.3 SegNet	65

4.6.3.1	<i>L'entraînement du modèle</i>	69
4.6.3.2	<i>Les résultats de l'entraînement</i>	69
4.6.3.3	<i>L'évaluation du modèle</i>	71
4.6.3.4	<i>Tester le modèle sur les images du test</i>	72
4.6.4	<i>UNet</i>	72
4.6.4.1	<i>L'entraînement du modèle</i>	73
4.6.4.2	<i>Les résultats de l'entraînement</i>	74
4.6.4.3	<i>L'évaluation du modèle</i>	76
4.6.5	<i>Attention-UNet</i>	78
4.6.5.1	<i>L'entraînement du modèle</i>	81
4.6.5.2	<i>Les résultats de l'entraînement</i>	82
4.6.5.3	<i>L'évaluation du modèle</i>	84
4.6.5.4	<i>Tester le modèle sur les images du test</i>	85
4.7	<i>Comparaison entre les modèles</i>	86
4.8	<i>Conclusion</i>	86
5	<i>Conclusion</i>	87

Table des figures

2.1 <i>Illustration du principe de la croissance de régions</i>	12
2.2 <i>Illustration du principe de Watershed</i>	14
3.1 <i>a) Segmentation d'image à l'aide du seuillage b) Segmentation sémantique</i>	22
3.2 <i>Inférence grossière à fine</i>	23
3.3 <i>Architecture FCN [1]</i>	24
3.4 <i>(en haut) convolution régulière suivie d'un suréchantillonnage, (en bas) convolution atreuse résultant d'une carte de caractéristiques plus dense. [4]</i>	26
3.5 <i>Organigramme Deeplabv1 [12]</i>	27
3.6 <i>Pooling de pyramides spatiales Atrous (ASPP)[13]</i>	27
3.7 <i>3 × 3 Convolution séparable en profondeur pour une convolution atreuse. [5]</i>	28
3.8 <i>Modèle Deeplabv3+. [11]</i>	29
3.9 <i>Comparaison entre FCN et PSPNet [1]</i>	29
3.10 <i>Réseaux Encodeur-Décodeur pour la Segmentation Sémantique</i>	30
3.11 <i>Convolution avec dilation</i>	31
3.12 <i>Architecture PSPNet</i>	31
3.13 <i>PSPNet avec un décodeur à suréchantillonnage 8x</i>	32
3.14 <i>Connexions de saut de l'encodeur au décodeur similaire à U-Net</i>	33
3.15 <i>Architecture U-Net</i>	35
3.16 <i>Architecture U-Net++</i>	37
4.1 <i>Images cérébrales segmentés manuellement</i>	43
4.2 <i>Intersection Over Union (IoU)</i>	46
4.3 <i>Représentation graphique d'IoU</i>	47

4.4 Performance comparative de l'indice de Jaccard (IoU)	47
4.5 Représentation graphique de Dice coefficient	48
4.6 Synthèse de la Performance du Modèle DeepLabV3+ : Données d'Entraînement et de Validation	57
4.7 Synthèse de la Performance du Modèle PSPNet : Données d'Entraînement et de Validation	63
4.8 Synthèse de la Performance du Modèle SegNet : Données d'Entraînement et de Validation	71
4.9 Synthèse de la Performance du Modèle Unet : Données d'Entraînement et de Validation	76
4.10 Synthèse de la performance du modèle Attention Unet : Données d'entraînement et de validation	84

1

Introduction

Préambule

L'identification des tumeurs cérébrales revêt une importance capitale en neurologie et en médecine pour assurer des diagnostics rapides et des traitements efficaces. Les avancées récentes dans le domaine de l'apprentissage en profondeur ont ouvert de nouvelles perspectives pour la détection automatisée et la segmentation précise des tumeurs cérébrales à partir d'images médicales.

1.1 Contexte du Projet

Les progrès technologiques, spécifiquement dans le domaine de l'apprentissage en profondeur, ont considérablement amélioré la détection et la segmentation des tumeurs cérébrales à partir d'images médicales, en particulier les IRM. Cette évolution représente une avancée majeure dans le domaine de la neurologie, offrant des outils pour une identification plus précise et rapide des pathologies cérébrales.[10]

1.2 Problématique

Malgré ces avancées notables, la segmentation automatique des tumeurs cérébrales présente toujours des défis majeurs. Les variations de taille, de forme et de localisation des tumeurs, ainsi que les différences dans les caractéristiques des images, complexifient cette tâche.

1.3 Objectif de l'Étude

Ce projet vise à explorer et à comparer plusieurs méthodes d'apprentissage en profondeur pour la segmentation et la détection des tumeurs cérébrales. L'objectif principal est d'évaluer la performance

de différents modèles dans leur capacité à identifier et localiser précisément les tumeurs cérébrales à partir d'images IRM.

1.4 Structure du Rapport

Ce rapport se divise en quatre chapitres principaux :

1. **Segmentation des IRM cérébraux dans la littérature :** Une analyse des méthodes existantes pour la segmentation des tumeurs cérébrales par apprentissage en profondeur.
2. **Approches de Segmentation pour la Détection des Tumeurs Cérébrales :** Description détaillée des modèles analysés, incluant U-Net, DeepLabV3+, etc., avec leurs architectures, avantages et limitations.
3. **Évaluation Comparative des Modèles de Segmentation :** Détails sur la mise en œuvre des modèles, la méthodologie d'évaluation comparative, les métriques utilisées et la présentation des résultats.
4. **Conclusion :** Synthèse des résultats obtenus, implications médicales et perspectives futures.

En analysant ces méthodes, ce projet aspire à contribuer aux outils diagnostiques pour une détection plus efficace des tumeurs cérébrales, ouvrant la voie à des soins plus précis et personnalisés pour les patients.

Segmentation des IRM cérébraux dans la littérature

Préambule

Dans cette partie, nous plongerons dans l'univers des techniques de segmentation des tumeurs cérébrales, telles qu'elles ont été décrites et expérimentées dans des études précédentes. L'objectif est de parcourir les diverses approches utilisées dans la recherche pour découper les images IRM du cerveau, permettant ainsi l'identification précise des différentes tumeurs cérébrales en s'appuyant sur les avancées de l'apprentissage en profondeur.

2.1 Introduction

2.1.1 Définition de la segmentation

La segmentation d'image, de manière générale, est un processus essentiel en traitement d'images, consistant à diviser une image en régions ou en pixels ayant des caractéristiques similaires. Cette division permet une analyse plus approfondie et précise de l'image. En utilisant diverses techniques et algorithmes, la segmentation peut être réalisée en fonction de la couleur, de la texture, de la luminosité, ou d'autres caractéristiques visuelles. Les méthodes de segmentation varient en fonction de la complexité de l'image et du contexte d'application. Parmi ces méthodes, on trouve la segmentation par seuillage, les méthodes de contour actif, la segmentation basée sur les régions, et les approches utilisant l'apprentissage automatique ou en profondeur.

Dans le domaine médical, la segmentation d'images, notamment dans les IRM cérébrales, revêt une importance cruciale pour diagnostiquer et évaluer des conditions telles que les tumeurs cérébrales. Les IRM génèrent des images détaillées du cerveau, mais leur analyse nécessite souvent une segmentation précise pour isoler les régions d'intérêt. En segmentant ces images, les cliniciens

peuvent extraire des informations sur la taille, la forme et la localisation des tumeurs, ainsi que sur d'autres structures anatomiques. Cela facilite le diagnostic différentiel, la planification chirurgicale, et la surveillance de l'évolution des tumeurs au fil du temps. Les techniques de segmentation spécifiques au domaine médical peuvent impliquer l'utilisation de réseaux de neurones convolutionnels (CNN) ou d'autres méthodes d'apprentissage en profondeur pour distinguer les tissus normaux des zones pathologiques dans les images médicales, améliorant ainsi la précision des diagnostics.

2.1.2 Les objectifs de la segmentation

La segmentation d'images poursuit plusieurs objectifs essentiels. Parmi ceux-ci :

Analyse et identification précise des régions d'intérêt : La segmentation permet d'isoler et d'identifier spécifiquement les régions d'intérêt dans une image. Cela peut inclure la détection de contours, la séparation de différentes structures anatomiques, ou la mise en évidence de zones pathologiques comme les tumeurs dans le domaine médical.

Amélioration de la compréhension et de l'interprétation : En divisant une image en parties distinctes, la segmentation facilite la compréhension visuelle et l'interprétation des informations présentes. Cela permet une analyse plus approfondie des données visuelles, souvent en fournissant des informations précieuses sur les structures et les caractéristiques visibles.

Prétraitement pour des applications ultérieures : La segmentation est souvent utilisée comme étape de prétraitement dans diverses applications, telles que la reconnaissance d'objets, la reconstruction 3D, ou l'analyse quantitative des images médicales. Une segmentation précise améliore la qualité et la fiabilité des résultats obtenus dans ces applications.

Facilitation de la prise de décision : Dans le domaine médical, une segmentation précise est cruciale pour aider les professionnels de la santé à prendre des décisions éclairées. En identifiant clairement les structures anatomiques ou les anomalies, elle contribue à des diagnostics plus précis et à la planification de traitements adaptés.

Automatisation des tâches : La segmentation permet également d'automatiser certaines tâches, notamment dans le cadre de l'analyse d'images médicales. Elle peut servir de base à des algorithmes d'apprentissage automatique pour la classification, la prédiction et l'analyse quantitative des données visuelles.

2.2 État de l'art du point de vue « Traitement d'images »

2.2.1 Segmentation basée sur les régions

La segmentation basée sur les régions est une approche courante qui vise à regrouper des pixels ou des voxels en régions homogènes selon certaines propriétés. Cela peut inclure des critères tels que la similarité de couleur, d'intensité ou de texture. Des techniques telles que la croissance de régions, les méthodes de watershed et les algorithmes de région-merging sont souvent utilisées pour cette approche. Elles visent à identifier des régions cohérentes et homogènes dans une image en se basant sur des critères prédéfinis.[17]

2.2.1.1 Méthodes de croissance de régions

Formulation mathématique :

La croissance de régions repose sur une condition de similarité entre les pixels, souvent définie comme la différence entre les niveaux de gris (ou les valeurs de pixel) des pixels adjacents. Soit P_i et P_j deux pixels adjacents dans une image I , la condition de similarité est généralement définie comme :

$$|I(P_i) - I(P_j)| \leq \text{Seuil}$$

Où $I(P_i)$ et $I(P_j)$ sont les intensités des pixels P_i et P_j respectivement, et le seuil est un paramètre prédéfini pour contrôler la similarité.



FIGURE 2.1: Illustration du principe de la croissance de régions

Algorithme croissance de régions

- 1: **Entrée :** Image I , pixel de départ (x_0, y_0)
- 2: **Sortie :** Région segmentée R
- 3: Initialiser la région R avec le pixel de départ (x_0, y_0)

- 4: Initialiser une liste vide Q pour stocker les pixels à vérifier
- 5: Ajouter les voisins de (x_0, y_0) à la liste Q
- 6: **while** Q n'est pas vide **do**
- 7: Retirer un pixel (x, y) de Q
- 8: **if** (x, y) satisfait au critère de similarité avec les pixels de R **then**
- 9: Ajouter (x, y) à la région R
- 10: Ajouter les voisins de (x, y) à la liste Q
- 11: **end if**
- 12: **end while**
- 13: **return** Région segmentée R

Avantages et Limitations

Avantages :

- **Facile à mettre en œuvre :** Cette méthode ne nécessite pas une complexité algorithmique élevée pour être implémentée.
- **Adapté pour des zones homogènes :** Elle fonctionne bien dans des zones où les pixels ont des valeurs similaires, donc des zones relativement homogènes dans l'image.

Limitations :

- **Sensible aux conditions initiales :** Les résultats de cette méthode peuvent varier en fonction du choix du pixel de départ. Selon l'emplacement initial sélectionné, la région résultante peut différer.
- **Difficulté avec des transitions subtiles :** Lorsque la différence entre les pixels voisins est subtile, cette méthode peut avoir du mal à distinguer la frontière entre deux régions, ce qui peut conduire à une sous-segmentation ou à une fusion de régions qui devraient être distinctes.

2.2.1.2 Méthode de Watershed

La méthode de Watershed est une technique basée sur la transformation du gradient de l'image. Elle est souvent utilisée pour la segmentation d'images, en particulier dans les cas où il existe des régions de transition et des zones mal délimitées.

Formulation Mathématique

La carte du gradient (G) d'une image bidimensionnelle $I(x, y)$ est obtenue en calculant le gradient

de l'image, souvent défini comme la magnitude du gradient :

$$G = \sqrt{G_x^2 + G_y^2}$$

où G_x et G_y sont les gradients en x et y respectivement.

Les lignes de partage des eaux sont déterminées comme les minima locaux de la carte du gradient G .

Dans l'exemple ci-dessous, deux cercles superposés doivent être séparés. Pour ce faire, on calcule une image qui représente la distance par rapport à l'arrière-plan. Les maxima de cette distance (c'est-à-dire les minima de l'opposé de la distance) sont choisis comme marqueurs, et l'inondation des bassins à partir de ces marqueurs sépare les deux cercles le long d'une ligne de partage des eaux.

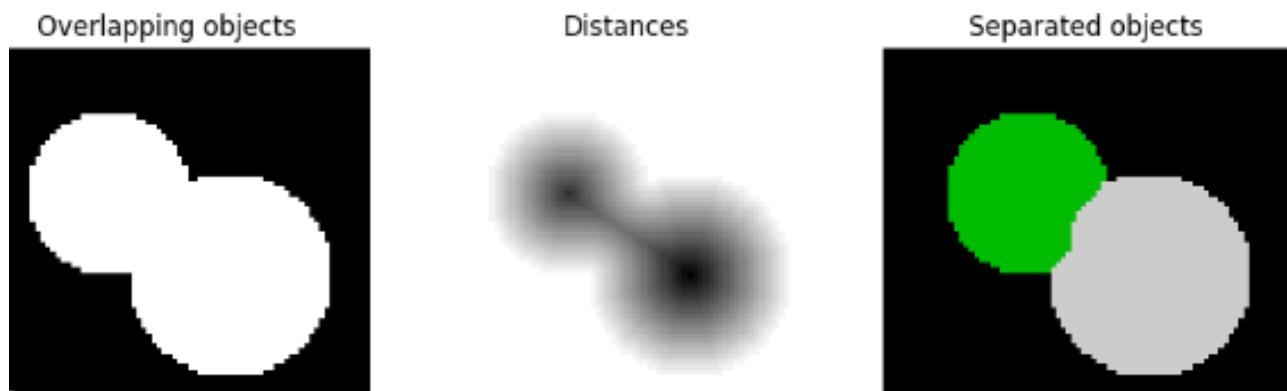


FIGURE 2.2: Illustration du principe de Watershed

Algorithme de Watershed

- 1: **Entrée :** Image I , Marqueurs
- 2: **Sortie :** Région segmentée R
- 3: Calculer la carte du gradient G de l'image I
- 4: Identifier les minima locaux de G comme marqueurs pour Watershed
- 5: Appliquer l'algorithme de Watershed à partir des marqueurs identifiés
- 6: Fusionner les régions en fonction des marqueurs
- 7: **return** Région segmentée R

Avantages et Limitations

Avantages :

- **Capable de détecter les détails fins :** Watershed est idéal pour détecter des détails fins dans les images et est bien adapté pour les transitions claires.

Limitations :

- **Peut sur-segmenter dans des zones homogènes :** Dans les zones où les variations de pixels sont subtiles, Watershed peut produire une sur-segmentation, générant plusieurs petites régions au lieu de regrouper les régions similaires.
- **Sensibilité aux conditions initiales :** Le choix initial des marqueurs peut grandement affecter le résultat de la segmentation, et des marqueurs inappropriés peuvent conduire à des régions mal segmentées.

2.2.1.3 Méthode de Région-Merging

La méthode de région-merging est une approche de segmentation d'image qui combine des régions similaires pour former des segments plus grands et significatifs. Elle se base sur le concept de fusion de régions similaires en utilisant des critères de similarité.

Formulation Mathématique

Soit R_i et R_j deux régions adjacentes ou voisines dans une image I . La condition de similarité entre ces régions peut être définie par divers critères, tels que la différence entre les niveaux de gris moyens des régions :

$$|Moy(R_i) - Moy(R_j)| \leq \text{Seuil de Similarité}$$

où $Moy(R_i)$ et $Moy(R_j)$ représentent les niveaux de gris moyens des régions R_i et R_j respectivement.

L'algorithme de région-merging fusionne les régions qui satisfont ce critère de similarité, formant ainsi des régions plus grandes et plus cohérentes.

Algorithme de Région-Merging

- 1: **Entrée :** Image I , Seuil de Similarité
- 2: **Sortie :** Régions segmentées R
- 3: Initialiser R avec des régions de taille unitaire pour chaque pixel de l'image I
- 4: **while** Il existe des régions qui satisfont le critère de similarité **do**
- 5: Fusionner les régions voisines qui satisfont le critère de similarité
- 6: **end while**
- 7: **return** Régions segmentées R

Avantages et Limitations

Avantages :

- **Conservation des détails locaux :** La méthode de région-merging conserve généralement les détails locaux des objets dans l'image.

Limitations :

- **Sensible aux variations de seuil :** La performance de cette méthode peut varier considérablement selon le seuil de similarité choisi, ce qui peut entraîner une sous-segmentation ou une sur-segmentation.
- **Complexité algorithmique :** Le processus de fusion de régions peut être coûteux en termes de temps de calcul pour de grandes images ou des critères de similarité complexes.

2.2.2 Segmentation basée sur les contours

Les approches basées sur les contours se distinguent des méthodes régionales en se concentrant sur les discontinuités présentes dans les images pour détecter les contours des différentes régions. Nous explorerons plus en détail les techniques dérivatives et les méthodes qui reposent sur des modèles déformables.[16]

2.2.2.1 Méthodes dérivatives

Les méthodes dérivatives constituent une approche simple pour détecter les ruptures et contours dans les images. Elles identifient les contours comme des points de forts gradients ou de dérivées secondes nulles. Parmi ces méthodes, on trouve les filtres de Roberts, de Sobel, ou de Prewitt pour le gradient, et le Laplacien pour les dérivées secondes.

Par exemple, l'opérateur de Marr-Hildreth est souvent utilisé pour produire des contours fermés. Dans une étude [20], une approximation de cet opérateur a été employée pour segmenter le cerveau en différentes régions. Cependant, cette méthode a montré des décalages de contours par rapport à la réalité anatomique, nécessitant une fermeture morphologique pour corriger ces écarts.

Les méthodes dérivatives, bien que rapides et ne nécessitant pas d'informations préalables, sont sensibles au bruit et à la dérive du champ radiofréquence. Elles peuvent produire de la sous-segmentation ou de la sur-segmentation, étant très dépendantes des contrastes entre les structures recherchées.

2.2.2.2 Modèles déformables

Les algorithmes basés sur les modèles déformables sont une évolution des méthodes de contours dynamiques. Ils visent à fournir des contours ou surfaces fermés en minimisant itérativement une

*fonctionnelle, intégrant des informations *a priori* sur la forme de l'objet à détecter à travers des forces externes et internes.*

*Par exemple, dans une étude [6], une surface dynamique discrète a permis la segmentation de l'hippocampe en introduisant une nouvelle force externe basée sur la recherche de minima locaux dans l'image pour une déformation précise du contour. Une autre étude [9] a proposé un schéma de segmentation de l'arachnoïde adapté aux images anisotropiques en utilisant un modèle contraint par des informations *a priori*.*

Cependant, les modèles déformables ont des limites : ils sont sensibles à l'initialisation et peuvent nécessiter des ajustements pour les différentes structures anatomiques. Bien qu'efficaces pour segmenter des structures anatomiques spécifiques, ces méthodes sont moins adaptées pour la segmentation automatique des zones tumorales, en raison de la variété des localisations des tumeurs et de leur hétérogénéité.

2.2.3 Limites et défis rencontrés avec ces méthodes

Les méthodes de segmentation basée sur les régions et celles basées sur les contours présentent plusieurs limites et défis :

2.2.3.1 Méthodes de régions

- **Sensibilité aux contrastes :** *Elles dépendent fortement des contrastes entre les structures recherchées, ce qui peut limiter leur efficacité dans les zones présentant des variations subtiles.*
- **Sur-segmentation et sous-segmentation :** *Elles ont tendance à produire des résultats avec des régions non désirées ou à regrouper plusieurs régions en une seule.*
- **Dépendance aux conditions initiales :** *Les résultats peuvent être influencés par les conditions initiales, ce qui rend parfois difficile leur application automatique et reproductible.*

2.2.3.2 Méthodes de contours

- **Sensibilité au bruit :** *Elles sont souvent affectées par le bruit présent dans les images, ce qui peut conduire à la détection de faux contours ou à des résultats imprécis.*
- **Défis dans les transitions de niveaux de gris :** *Les contours peuvent être mal définis dans les zones où les variations de niveaux de gris sont subtiles, rendant difficile la détection précise des contours.*

- **Limites dans la détection des contours fermés :** Certaines méthodes peuvent avoir des difficultés à obtenir des contours fermés et précis, surtout dans des cas complexes.

Face à ces défis et limites des méthodes traditionnelles, une réflexion s'impose : Comment peut-on surmonter ces limitations pour obtenir des résultats plus précis et robustes dans la segmentation d'images médicales ?

2.3 État de l'art du point de vue « Reconnaissance des formes »

Les approches de segmentation discutées précédemment s'inscrivent dans le contexte d'une analyse purement "image". Ces méthodes visent à identifier des régions ou des contours. Cependant, un autre angle d'approche important est celui de la reconnaissance des formes, qui aborde le problème de la prise de décision dans des scénarios de classification [18]. De manière plus rigoureuse, ce problème implique la classification d'individus, représentés par un ensemble de caractéristiques, parmi un ensemble de classes préétablies, définies ou non. Pour traiter un problème de reconnaissance des formes [15], il est nécessaire de :

- *Définir les paramètres constituant le vecteur forme x , représentatif de l'état du système; la dimension de x correspond à celle de l'espace des caractéristiques.*
- *Établir l'ensemble des états ou classes connus pour lesquels des informations sont disponibles : modèles de comportement probabilistes, ensembles de vecteurs d'échantillons, etc.*
- *Élaborer une règle de décision qui associe à un vecteur forme x soit la décision d'assigner une classe, soit la décision de rejeter toutes les classes connues, soit l'absence de décision.*

Dans le contexte de la segmentation des images IRM, le vecteur forme x correspond aux niveaux radiométriques du point étudié. La taille de ce vecteur correspond donc au nombre de pondérations utilisées pour la segmentation. Cette formalisation s'adapte bien aux traitements multi-échos.

Parmi les approches de reconnaissance des formes, on distingue les méthodes supervisées, où les caractéristiques des classes sont préalablement connues, des méthodes non supervisées, qui apprennent ces caractéristiques par elles-mêmes. Les termes "apprentissage supervisé" et "apprentissage non supervisé" sont respectivement associés aux concepts de "classement" ou "discrimination", et de "classification".

Certaines de ces méthodes intègrent des informations contextuelles, les positionnant ainsi dans un processus de segmentation plutôt que de simple discrimination ou classification.

2.3.1 Transition vers l'Apprentissage en Profondeur

L'apprentissage en profondeur a révolutionné la segmentation d'images IRM en introduisant des approches basées sur des réseaux de neurones profonds. Cette transition a marqué un changement significatif dans la manière dont la segmentation est abordée, offrant de nouvelles perspectives et des résultats prometteurs.

2.3.1.1 Définition de l'Apprentissage en Profondeur

L'apprentissage en profondeur fait référence à un ensemble de techniques d'apprentissage automatique où les réseaux de neurones profonds sont utilisés pour extraire des caractéristiques hiérarchiques à partir des données. Ces réseaux, avec leurs multiples couches de traitement, peuvent apprendre des représentations de données complexes.

2.3.1.2 Utilisation de l'Apprentissage Profond dans la Segmentation

Les méthodes d'apprentissage en profondeur ont été largement adoptées pour la segmentation d'images IRM en raison de leur capacité à apprendre des représentations de haut niveau. Des architectures comme U-Net, SegNet, et DeepLab sont utilisées pour apprendre des représentations des données médicales et effectuer la segmentation pixel par pixel.

2.3.2 Techniques Actuelles de Segmentation avec les Réseaux de Neurones

1. **U-Net** : Architecture encodeur-décodeur utilisée pour la segmentation sémantique. Elle conserve les informations spatiales via des connexions résiduelles entre les couches.

2. **DeepLab** : Utilise des CNN avec des mécanismes d'attribution de poids aux pixels pour améliorer la précision des contours en utilisant des dilations convolutives et des blocs atrous.

3. **SegNet** : Architecture encodeur-décodeur pour extraire des caractéristiques et reconstruire l'image segmentée, utilisant des cartes de pooling inversées pour la reconstruction.

4. **FCN (Fully Convolutional Network)** : Utilise exclusivement des couches convolutives pour capturer des informations spatiales à différentes échelles et produire des cartes de segmentation.

5. **Mask R-CNN** : Une extension du modèle Faster R-CNN qui combine la détection d'objets avec une segmentation précise des instances.

6. **PSPNet (Pyramid Scene Parsing Network)** : Utilise des pyramides de pooling pour capturer des informations contextuelles à différentes échelles spatiales.

7. **HRNet (High-Resolution Network)** : Conçu pour maintenir des résolutions spatiales élevées tout au long du réseau pour une segmentation précise à des échelles différentes.

8. **UNet++** : Une amélioration de U-Net avec des chemins résiduels connectés pour une meilleure représentation des caractéristiques.

9. **Attention U-Net** : Intègre des mécanismes d'attention pour se concentrer sur des parties spécifiques de l'image lors de la segmentation.

10. **LinkNet** : Utilise des connexions de type "link" pour établir des liens entre différentes couches du réseau et améliorer la performance de segmentation.

Chaque méthode présente des variations dans son architecture et son approche de la segmentation, avec des avantages spécifiques pour différents types de données et d'applications médicales.

2.3.3 Avantages et Limitations

Avantages : L'apprentissage en profondeur offre une meilleure capacité d'apprentissage des caractéristiques complexes, une généralisation améliorée sur des données inconnues, et des performances remarquables pour la segmentation précise des structures anatomiques.

Limitations : Ces méthodes peuvent nécessiter des volumes de données étiquetées importants, des ressources computationnelles élevées et leur opacité peut limiter leur interprétabilité.

2.3.4 Conclusion

Cette transition vers l'apprentissage en profondeur a révolutionné la segmentation d'images médicales en fournissant des méthodes plus précises et adaptatives, bien que certains défis persistent.

Approches de Segmentation pour la Détection des Tumeurs Cérébrales

_____ Préambule _____

Ce chapitre se concentre sur les différentes approches de segmentation utilisées spécifiquement pour détecter les tumeurs cérébrales à partir d'images médicales. Nous explorerons en détail diverses méthodes de segmentation en apprentissage profond, en mettant l'accent sur leur fonctionnement, leurs avantages et leurs limitations.

3.1 Introduction

3.1.1 Segmentation d'image et segmentation sémantique

En vision par ordinateur, la segmentation d'image simple est le processus de partitionnement d'une image numérique en plusieurs segments (ensembles de pixels). La segmentation d'image est un problème de vision par ordinateur ancien. Cependant, la segmentation sémantique est la technique de segmentation d'image avec une "compréhension" de l'image au niveau des pixels. En d'autres termes, la segmentation sémantique est l'analyse et la classification de chaque pixel en plusieurs classes (étiquettes).

Il existe de nombreuses applications de la segmentation sémantique, notamment la conduite autonome, l'interaction homme-machine, la photographie computationnelle, les moteurs de recherche d'images et la réalité augmentée, pour n'en citer que quelques-unes.

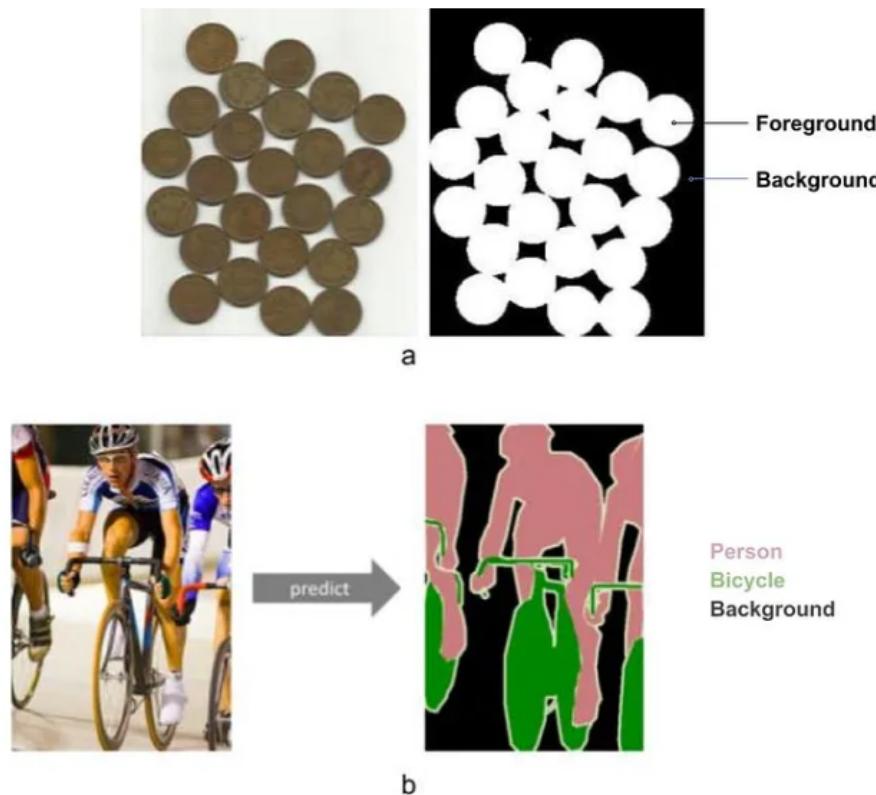


FIGURE 3.1: a) Segmentation d'image à l'aide du seuillage b) Segmentation sémantique

3.1.2 Historique de la segmentation sémantique

De nombreux algorithmes ont été conçus pour résoudre cette tâche, tels que l'algorithme Watershed, le seuillage d'image, le regroupement K-means, les méthodes de partitionnement de graphes, etc. La méthode la plus simple serait la méthode de seuillage. Dans cette méthode, une image en niveaux de gris est convertie en une image binaire en fonction d'une valeur de seuil. Malgré de nombreuses techniques traditionnelles de traitement d'image, les méthodes d'apprentissage profond ont été le facteur déterminant. Pour bien comprendre comment la segmentation sémantique est abordée par les architectures modernes d'apprentissage en profondeur, il est important de savoir qu'il ne s'agit pas d'un domaine isolé. Il s'agit plutôt d'une étape naturelle dans la progression de l'inférence grossière à fine. L'origine peut être située au niveau de la classification, qui consiste à prédire un objet entier à partir d'une image. L'étape suivante vers l'inférence fine est la localisation ou la détection, fournissant non seulement les classes mais également des informations supplémentaires concernant l'emplacement spatial de ces classes. Une segmentation sémantique peut être considérée comme une tâche de prédiction dense. Dans la prédiction dense, l'objectif est de générer une carte de sortie de la même taille que celle de l'image d'entrée. Il est évident que la segmentation sémantique est l'étape naturelle pour atteindre une inférence fine. Son objectif est de réaliser des prédictions denses en attribuant des étiquettes pour chaque pixel. Ainsi, chaque pixel est étiqueté avec la classe de

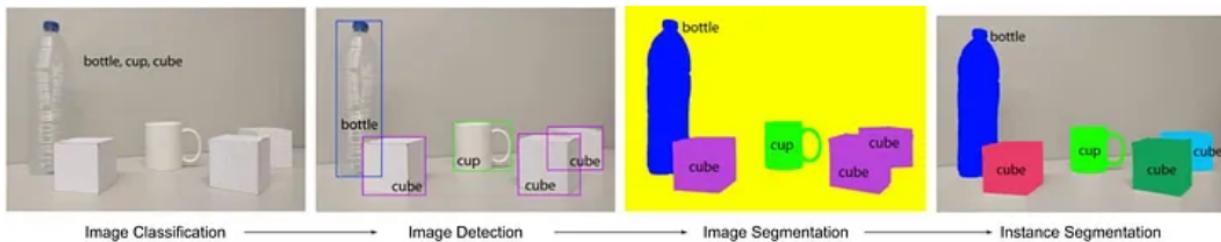


FIGURE 3.2: Inférence grossière à fine

son objet ou région englobant. Des améliorations supplémentaires peuvent être apportées, telles que la segmentation par instance (étiquettes distinctes pour différentes instances de la même classe).

Une architecture générale de segmentation sémantique peut être largement considérée comme un réseau d'encodeurs suivi d'un réseau de décodeurs :

- *L'encodeur est souvent constitué d'un réseau pré-entraîné pour la classification tel que VGG ou ResNet, suivi par un réseau de décodeurs.*
- *Le décodeur projette les caractéristiques discriminatives apprises par l'encodeur (à basse résolution) dans l'espace des pixels pour obtenir une classification dense (à haute résolution).*

3.1.3 Réseaux de Neurones Convolutifs Profonds dans la Segmentation

Les réseaux de neurones convolutifs profonds (DCNN) ont prouvé leur efficacité dans diverses tâches de vision par ordinateur. Cependant, leur application directe à la segmentation présente des limitations spécifiques liées à leur architecture et à leur fonctionnement.

3.1.3.1 Limitations des DCNN dans la Segmentation

Les réseaux de neurones convolutifs profonds (DCNN) ont montré des performances exceptionnelles dans diverses tâches de vision par ordinateur. Cependant, leur utilisation directe pour la segmentation présente des limitations spécifiques dues à leur architecture et à leur fonctionnement.

Limites :

- **Baisse de résolution du signal :** Les DCNN utilisent des techniques telles que le max-pooling et le downsampling pour extraire des caractéristiques. Cela réduit la résolution spatiale des cartes d'activation, impactant la précision spatiale nécessaire pour la segmentation détaillée.
- **Insensibilité spatiale :** Les DCNN sont conçus pour être invariants aux transformations spatiales pour obtenir des décisions centrées sur les objets. Cependant, cette invariance limite la

précision spatiale requise pour des tâches de segmentation sémantique où la localisation précise des contours est cruciale.

Avantages :

Bien que présentant des limitations pour la segmentation, les DCNN ont également des avantages significatifs :

- **Performances élevées :** Les DCNN ont considérablement amélioré les performances dans des domaines tels que la classification d'images, la détection d'objets, etc.
- **Invariance intégrée :** Leur capacité à apprendre des abstractions hiérarchiques des données en étant invariants aux transformations locales des images est précieuse pour des tâches de haut niveau.

L'un des tous premiers réseaux neuronaux convolutifs profonds (DCNN) utilisé pour la segmentation sémantique est le réseau Fully Convolutional Network (FCN). Le pipeline du réseau FCN est une extension du CNN classique. L'idée principale est de permettre au CNN classique de prendre en entrée des images de taille arbitraire. La limitation des CNN à accepter et produire des étiquettes uniquement pour des entrées de taille spécifique provient des couches entièrement connectées qui sont fixes. Contrairement à celles-ci, les FCN n'ont que des couches convolutionnelles et de pooling, ce qui leur donne la capacité de faire des prédictions sur des entrées de taille arbitraire. Un pro-

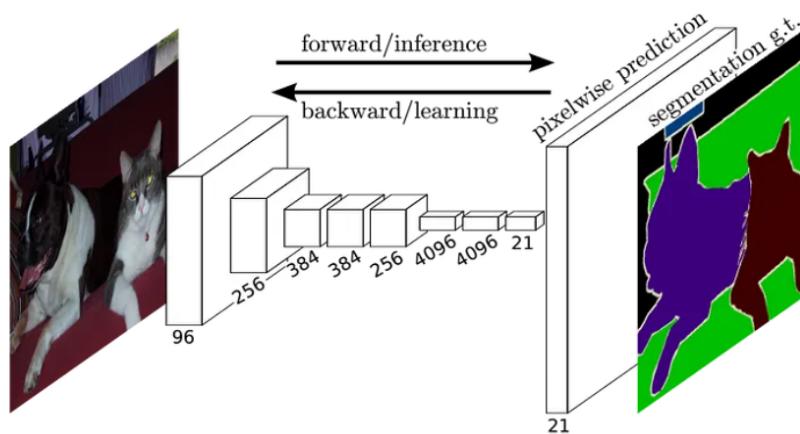


FIGURE 3.3: Architecture FCN [1]

blème spécifique de ce FCN est que, en se propageant à travers plusieurs couches convolutionnelles et de pooling alternées, la résolution des cartes de caractéristiques de sortie est échantillonnée vers le bas. Par conséquent, les prédictions directes du FCN sont généralement de faible résolution, ce qui entraîne des contours d'objets relativement flous. Une variété d'approches basées sur FCN plus

avancées ont été proposées pour résoudre ce problème, notamment SegNet, UNet, DeepLab et les convolutions dilatées.

3.1.4 Choix des Algorithmes pour Notre Étude

Notre choix s'est orienté vers plusieurs architectures renommées, chacune offrant des caractéristiques distinctes pour la segmentation des tumeurs cérébrales. DeeplabV3+, connu pour sa capacité à capturer des détails fins dans les images, est sélectionné pour sa précision dans la délimitation des contours. U-Net et sa variante Attention U-Net sont retenus pour leur architecture d'encodeur-décodeur qui maintient les informations spatiales tout en réduisant le sur-apprentissage. PSPNet est choisi pour son mécanisme de pyramide de contexte qui améliore la compréhension contextuelle des images. Enfin, SegNet est inclus pour sa capacité à utiliser des architectures légères tout en préservant la précision de la segmentation.[8]

3.2 Segmentation à l'aide de l'Architecture DeeplabV3+

3.2.1 Évolution de Deeplab

DeepLab est un modèle de segmentation sémantique de pointe conçu et mis en open-source par Google. La prédiction dense est obtenue en redimensionnant simplement la sortie de la dernière couche de convolution et en calculant la perte pixel par pixel. Deeplab applique une convolution atrous pour le redimensionnement.

3.2.2 Convolution Atrous

La combinaison répétée du max-pooling et du striding à des couches consécutives dans les DCNN réduit significativement la résolution spatiale des cartes de caractéristiques résultantes. Une solution consiste à utiliser des couches de déconvolution pour redimensionner la carte résultante. Cependant, cela nécessite de la mémoire et du temps supplémentaires. La convolution atrous offre une alternative simple et puissante à l'utilisation de la déconvolution. Elle permet d'agrandir efficacement le champ de vision des filtres sans augmenter le nombre de paramètres ou la quantité de calcul.

Mathématiquement, la convolution atrous $y[i]$ pour des signaux unidimensionnels $x[i]$ avec un filtre $w[k]$ de longueur K et un taux de pas r est définie comme suit :

$$y[i] = \sum_{k=0}^{K-1} x[i + r \cdot k] \times w[k]$$

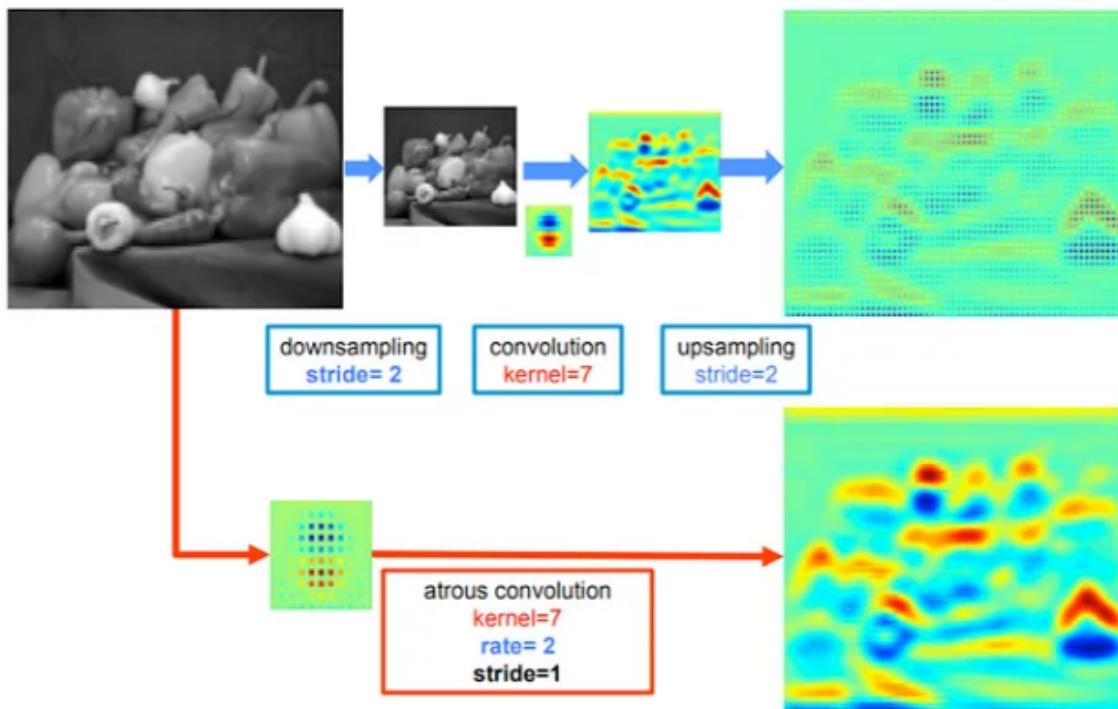


FIGURE 3.4: (en haut) convolution régulière suivie d'un suréchantillonnage, (en bas) convolution atreuse résultant d'une carte de caractéristiques plus dense. [4]

3.2.3 Deeplabv1

Le succès de Deeplabv1 dans la segmentation sémantique est dû à certaines avancées ajoutées aux modèles d'état de l'art précédents, spécifiquement au modèle FCN. Ces avancées adressent les deux défis suivants :

Défi 1 : réduction de la résolution des caractéristiques

En raison de multiples opérations de pooling et de sous-échantillonnage ('stride') dans les DCNN, il y a une réduction significative de la résolution spatiale. Ils suppriment l'opérateur de sous-échantillonnage des dernières couches de max-pooling des DCNN et redimensionnent plutôt les filtres (atrous) dans les couches de convolution suivantes, résultant en des cartes de caractéristiques calculées à un taux d'échantillonnage plus élevé.

Défi 2 : précision de la localisation réduite due à l'invariance des DCNN

Pour capturer les détails fins, ils utilisent un champ aléatoire conditionnel entièrement connecté (CRF). Les potentiels CRF intègrent des termes de régularisation qui maximisent l'accord des étiquettes entre les pixels similaires et peuvent intégrer des termes plus élaborés qui modélisent les relations contextuelles entre les classes d'objets.

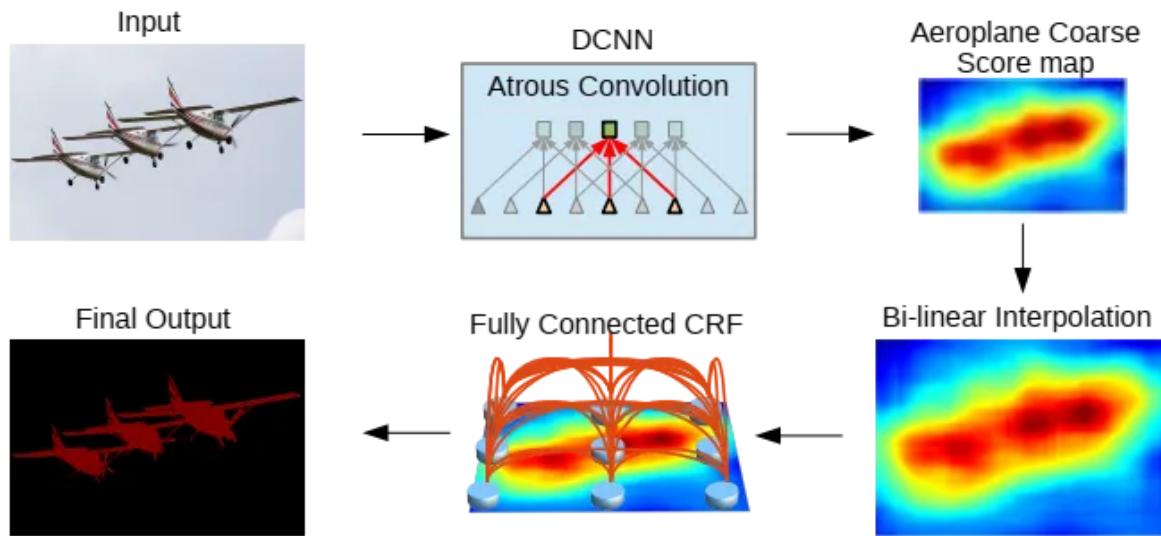


FIGURE 3.5: Organigramme Deeplabv1 [12]

3.2.4 Deeplabv2

Pour améliorer davantage les performances de l'architecture Deeplabv1, le défi suivant est l'*existence d'objets à plusieurs échelles*.

Défi : existence d'objets à plusieurs échelles

Pour représenter l'objet à plusieurs échelles, une méthode courante est de présenter aux DCNN des versions redimensionnées de la même image, puis d'agréger les caractéristiques ou cartes de score. **Solution : Utilisation du « Atrous Spatial Pyramid Pooling » (ASPP)**. L'idée est d'appliquer

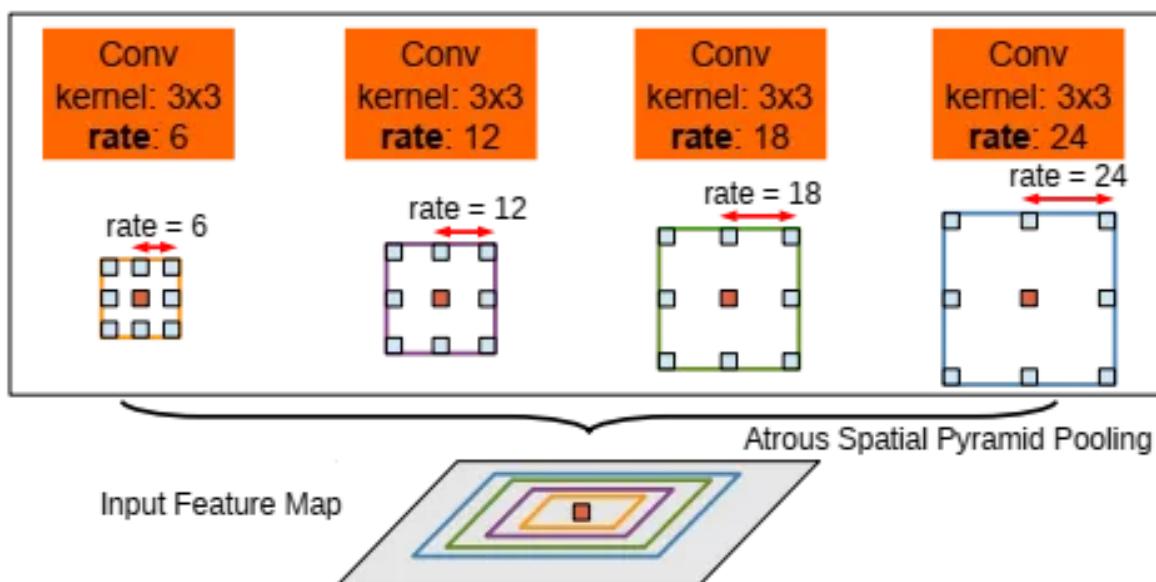


FIGURE 3.6: Pooling de pyramides spatiales Atrous (ASPP)[13]

plusieurs convolutions atrous avec différents taux d'échantillonnage à la carte de caractéristiques d'entrée et de les fusionner ensemble.

3.2.5 Deeplabv3

Les anciens réseaux sont capables d'encoder des informations contextuelles multi-échelles en sondant les caractéristiques entrantes avec des filtres ou des opérations de pooling (convolution atrous) à plusieurs taux et champs de vision efficaces (ASPP). Le défi suivant était de capturer des contours d'objets plus nets en récupérant progressivement les informations spatiales.

Défi : capturer des contours d'objets plus nets

L'architecture Deeplabv3 adopte un encodeur-décodeur novateur avec une convolution atrous séparable pour résoudre le problème ci-dessus. Le modèle encodeur-décodeur peut obtenir des contours d'objets nets.

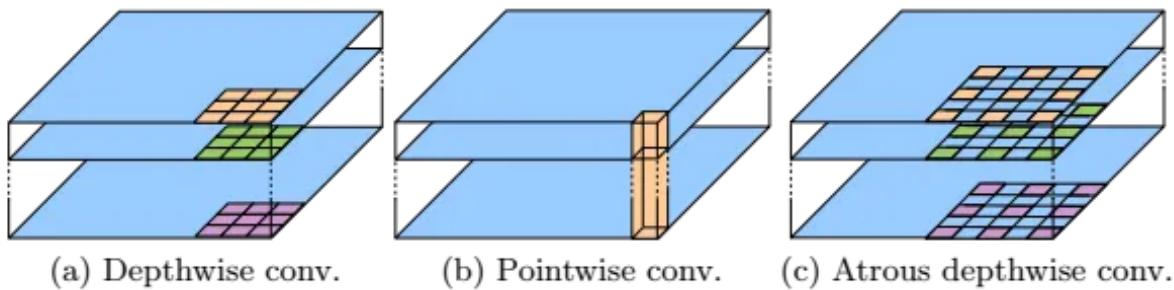


FIGURE 3.7: 3×3 Convolution séparable en profondeur pour une convolution atrouse. [5]

3.2.6 Deeplabv3+

Deeplabv3+ étend Deeplabv3 en ajoutant un module de décodeur simple mais efficace pour affiner davantage les résultats de segmentation, en particulier le long des contours d'objets.

Encodeur

Par rapport à Deeplabv3, il utilise Aligned Xception au lieu de ResNet-101 comme extracteur de caractéristiques principal (encodeur), mais avec une modification significative. Toutes les opérations de max-pooling sont remplacées par une convolution séparable en profondeur.

Décodeur

L'encodeur est basé sur une foulée de sortie de 16, c'est-à-dire que l'image d'entrée est échantillonnée à un facteur de 16. Ainsi, au lieu d'utiliser une interpolation bilinéaire avec un facteur de 16, les caractéristiques encodées sont d'abord échantillonnées à un facteur de 4 et concaténées

3.3. SEGMENTATION À L'AIDE DE L'ARCHITECTURE PSPNET

CÉRÉBRALES

avec les caractéristiques de niveau inférieur correspondantes du module encodeur ayant les mêmes dimensions spatiales. Après la concaténation, quelques convolutions 3×3 sont appliquées et les caractéristiques sont échantillonnées à un facteur de 4. Cela donne une sortie de la même taille que celle de l'image d'entrée.

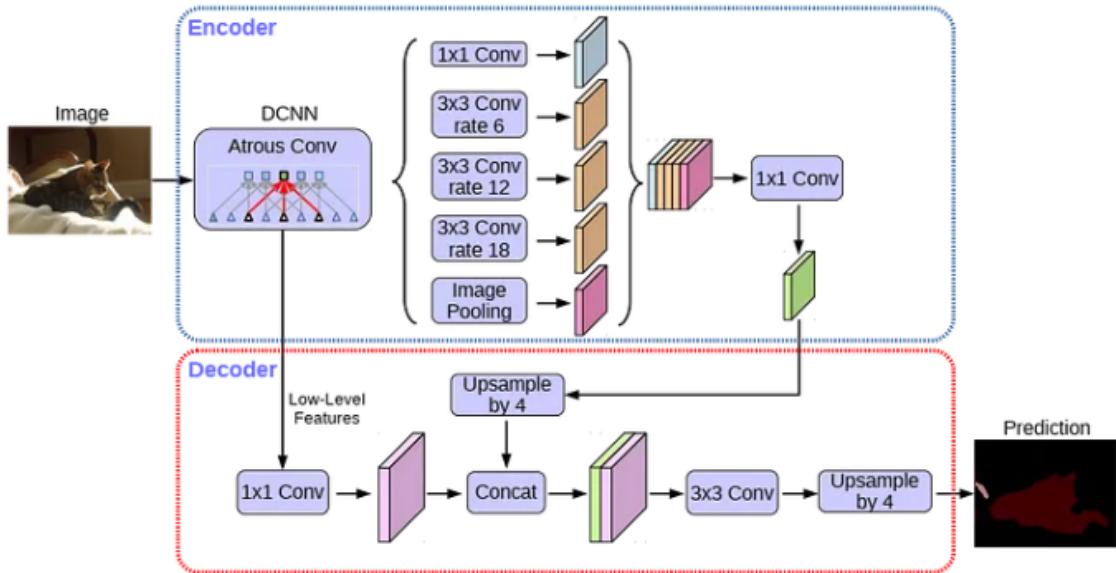


FIGURE 3.8: Modèle DeepLabv3+. [11]

3.3 Segmentation à l'aide de l'Architecture PSPNet

L'architecture PSPNet prend en compte le contexte global de l'image pour prédire les prédictions de niveau local, offrant ainsi de meilleures performances sur des ensembles de données de référence tels que PASCAL VOC 2012 et cityscapes. Ce modèle était nécessaire car les classificateurs de pixels basés sur FCN ne parvenaient pas à capturer le contexte de l'image entière.[8]

Exemple illustratif:

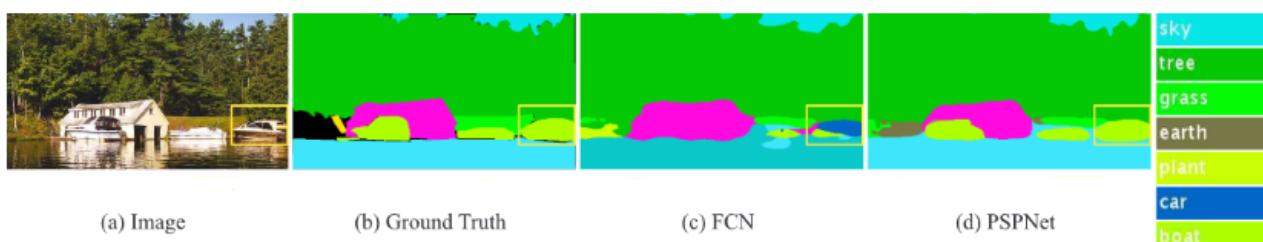


FIGURE 3.9: Comparaison entre FCN et PSPNet [1]

Le bateau à l'intérieur du cadre jaune dans (a) est classé comme une "voiture" par un classifica-

teur basé sur FCN (c), en raison de sa forme et de son apparence. Mais il est inhabituel de voir une voiture dans une rivière. Si le modèle pouvait obtenir des informations sur le contexte, par exemple dans notre cas l'eau autour de l'objet, il serait capable de le classer correctement. Le modèle PSPNet (d) est capable de capturer le contexte de l'image entière pour classer l'objet comme un bateau (en vert).

3.3.1 Architecture Encodeur-Décodeur pour la Segmentation Sémantique

La plupart des modèles de segmentation sémantique se composent de deux parties : un *Encodeur* et un *Décodeur*. L'*Encodeur* est responsable de l'extraction des caractéristiques de l'image, tandis que le *Décodeur* prédit la classe du pixel à la fin. Un *Encodeur-Décodeur* typique pour la tâche de segmentation ressemble à l'architecture illustrée ci-dessous :

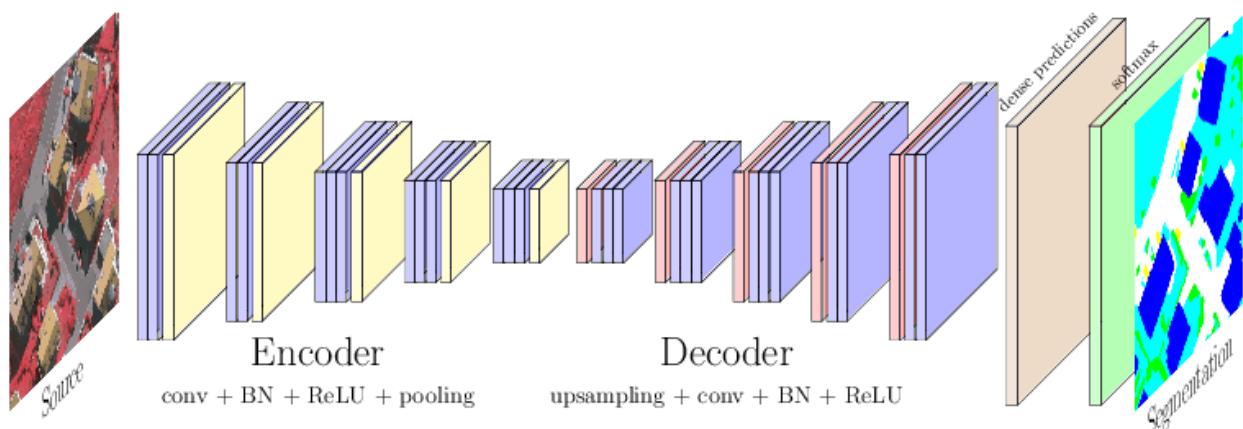


FIGURE 3.10: Réseaux Encodeur-Décodeur pour la Segmentation Sémantique

3.3.1.1 Encodeur PSPNet

L'*encodeur PSPNet* contient le réseau de neurones convolutifs (CNN) avec des *convolutions dilatées* ainsi que le module de regroupement en pyramide.

Convolutions Dilatées

Dans les dernières couches du réseau, nous remplaçons les couches de convolution traditionnelles par des couches de convolution dilatée, ce qui aide à augmenter le champ réceptif. Ces couches de convolution dilatée sont placées dans les deux derniers blocs du réseau. Ainsi, les caractéristiques obtenues à la fin du réseau contiennent des informations plus riches. L'illustration [2] montre comment fonctionnent les convolutions dilatées et en quoi elles diffèrent des convolutions traditionnelles.

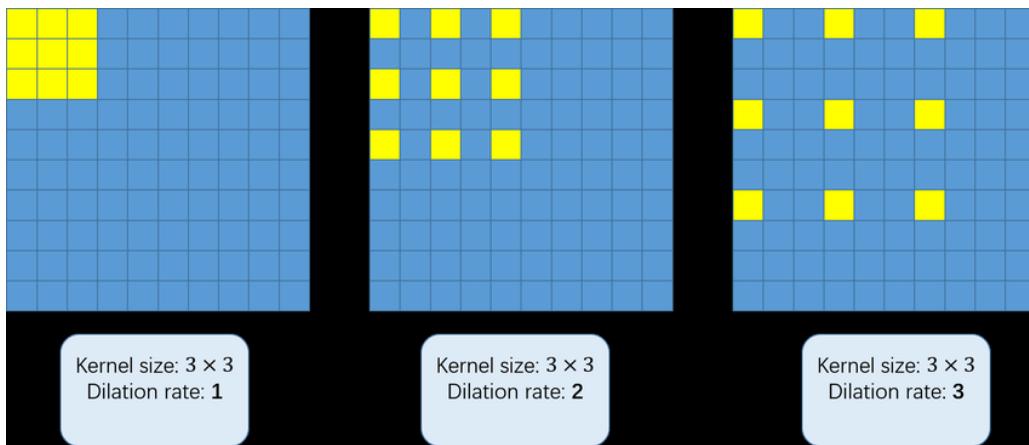


FIGURE 3.11: Convolution avec dilation

La valeur de dilation spécifie la densité lors de la convolution. On peut observer que le champ réceptif pour la convolution dilatée est plus grand par rapport à la convolution standard. La taille du champ réceptif indique quelle quantité d'informations contextuelles nous utilisons. Dans PSPNet, les deux derniers blocs du réseau ont des valeurs de dilation de 2 et 4 respectivement.

Module de Regroupement en Pyramide

Le module de regroupement en pyramide est la partie principale de ce modèle car il aide le modèle à capturer le contexte global dans l'image, ce qui l'aide à classifier les pixels en fonction des informations globales présentes dans l'image.

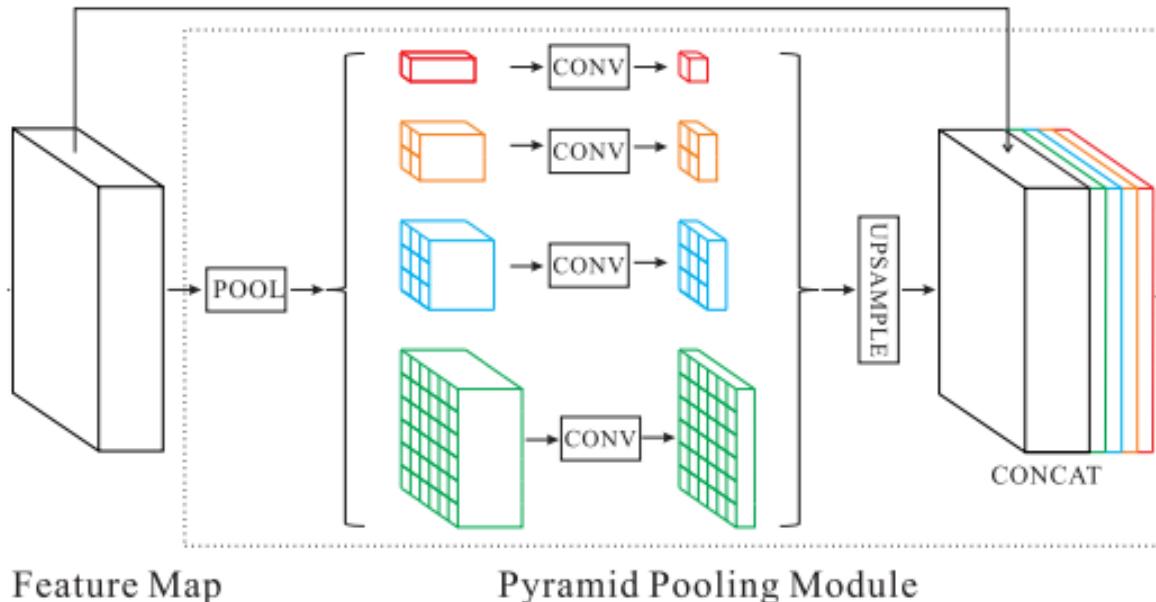


FIGURE 3.12: Architecture PSPNet

La carte de caractéristiques du réseau est regroupée à différentes tailles, puis passe à travers une couche de convolution, après quoi un suréchantillonnage est effectué sur les caractéristiques regrou-

pées pour les ramener à la même taille que la carte de caractéristiques d'origine. Enfin, les cartes suréchantillonnées sont concaténées avec la carte de caractéristiques d'origine pour être transmises au décodeur. Cette technique fusionne les caractéristiques de différentes échelles, agrémentant ainsi le contexte global.

3.3.1.2 Décodeur PSPNet

Après que l'encodeur a extrait les caractéristiques de l'image, c'est au tour du décodeur de prendre ces caractéristiques et de les convertir en prédictions en les passant à travers ses couches. Le décodeur est simplement un autre réseau qui prend en entrée des caractéristiques et produit des prédictions.

Décodeur à suréchantillonnage 8x

Le modèle PSPNet n'est pas un modèle de segmentation complet en soi, il s'agit simplement d'un encodeur, ce qui signifie qu'il représente seulement la moitié de ce qui est nécessaire pour la segmentation d'image. Les décodeurs les plus courants dans diverses implémentations de PSPNet sont une couche de convolution suivie d'un suréchantillonnage bilinéaire 8x.

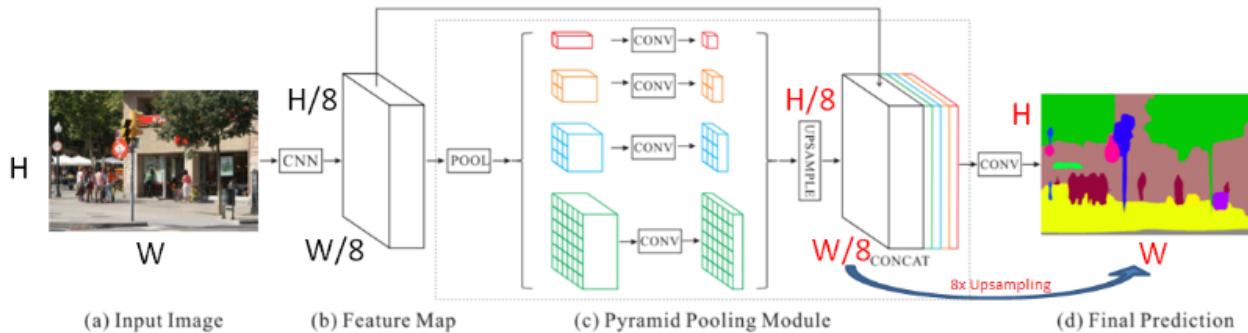


FIGURE 3.13: PSPNet avec un décodeur à suréchantillonnage 8x

Il y a un inconvénient à avoir un décodeur à suréchantillonnage 8x à la fin, car il n'y a pas de paramètres apprenants, les résultats obtenus sont donc flous et le modèle échoue à capturer des informations de haute résolution à partir de l'image.

PSPNet avec un décodeur similaire à U-Net

Maintenant, pour obtenir une sortie de haute résolution du modèle, une méthode consiste à avoir un décodeur qui possède des paramètres apprenants et qui peut prendre en compte des caractéristiques intermédiaires de l'encodeur en entrée. Pour y parvenir, nous nous sommes tournés vers le décodeur du réseau de pyramide de caractéristiques (FPN), qui est également utilisé dans U-Net. Ainsi, nous avons ajouté le décodeur FPN à l'encodeur PSPNet, capable de capturer les petites caractéristiques

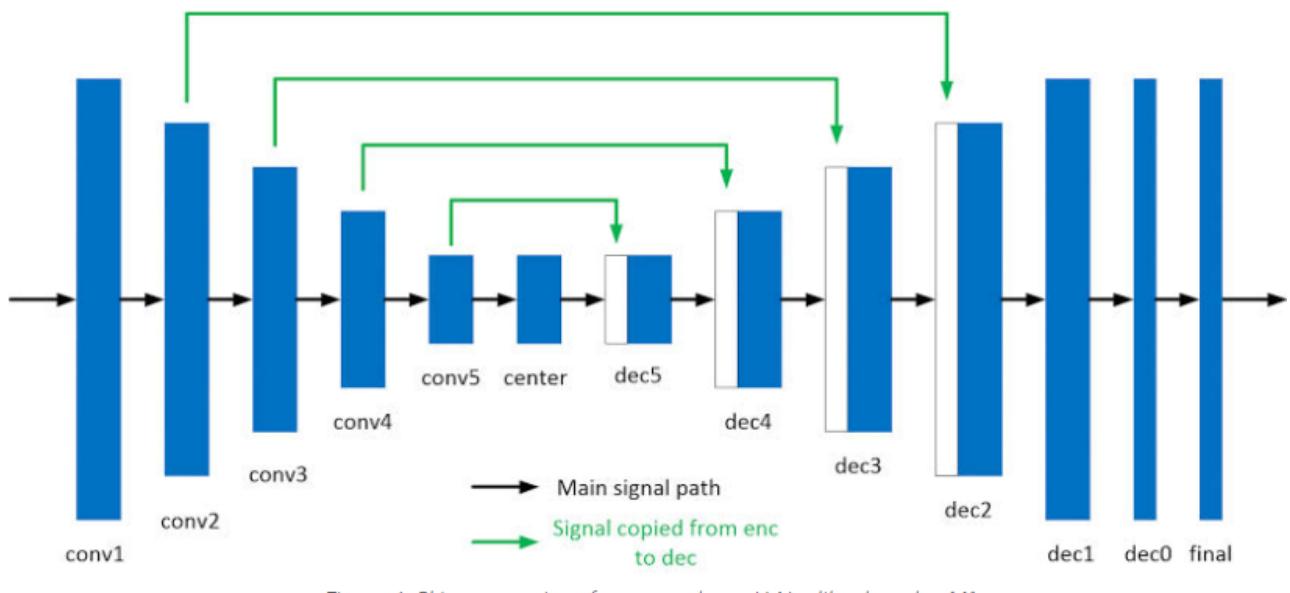
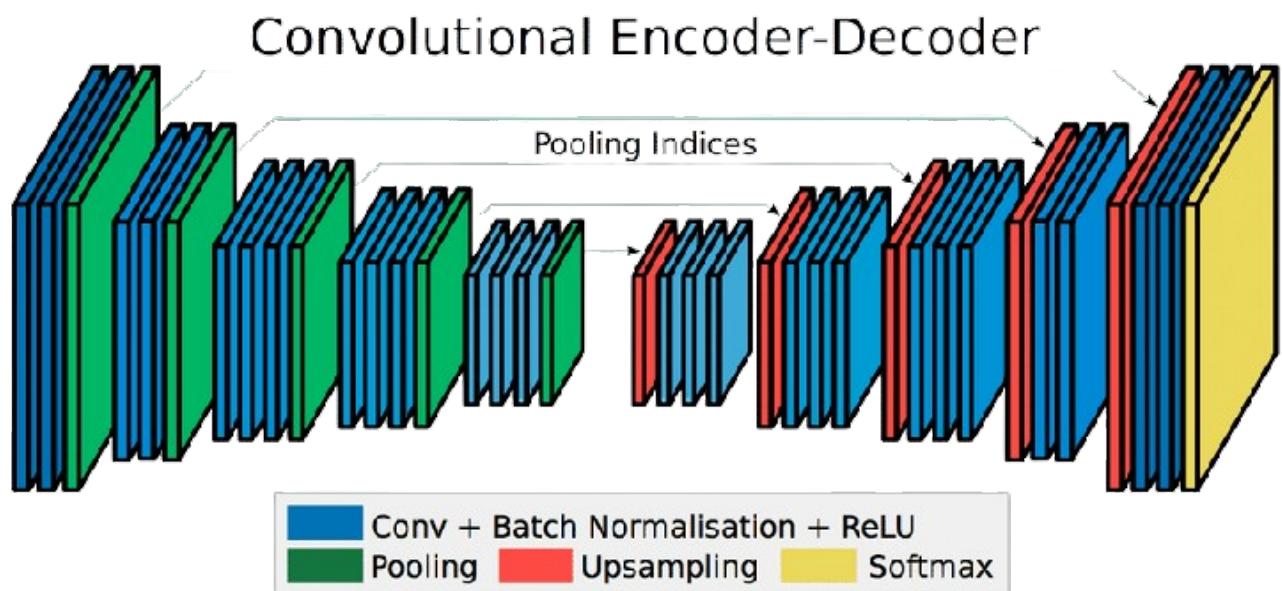


FIGURE 3.14: Connexions de saut de l'encodeur au décodeur similaire à U-Net

de l'image.

3.4 Segmentation à l'aide de l'Architecture SegNet

Segnet est l'abréviation de Segmentation Network, est une architecture de réseau neuronal convolutionnel (CNN) spécifiquement conçue pour la tâche de segmentation sémantique, qui consiste à attribuer une étiquette sémantique à chaque pixel de l'image. Son mécanisme opérationnel se fonde sur le principe encodeur-décodeur, structuré en différentes étapes clés.[2]



Encodage (Encoder)

Durant la phase d'encodage, les couches convolutionnelles de l'encodeur réduisent progressivement la résolution spatiale de l'image tout en préservant les informations sémantiques cruciales. Une particularité de SegNet réside dans l'utilisation de blocs de convolution spécifiques, les "Max-Pooling Indices", qui enregistrent les positions des valeurs maximales plutôt que les valeurs elles-mêmes, préservant ainsi des informations de localisation essentielles pour le décodage ultérieur. Cette approche vise à améliorer l'efficacité et la précision de la segmentation en conservant des détails spatiaux importants tout au long du processus.

Décodeur (Decoder)

Dans la phase de décodage, l'objectif essentiel est de reconstruire la segmentation sémantique à partir des caractéristiques abstraites extraites lors de l'encodage. Les indices enregistrés durant l'encodage sont exploités pour réaliser un "unpooling", consistant à insérer des valeurs nulles aux emplacements où les valeurs maximales ont été enregistrées au cours de la phase de max-pooling, préservant ainsi l'information spatiale cruciale. Cette étape est suivie par des opérations de déconvolution ou de transposition de convolution qui s'emploient à accroître graduellement la résolution spatiale de la carte de caractéristiques, permettant une reconstruction minutieuse et précise de la segmentation sémantique à partir des informations abstraites préalablement extraites.

Convolution en Profondeur (Depthwise Convolution)

Est une technique spécifique utilisée dans les réseaux neuronaux pour le traitement d'image. Elle divise l'opération de convolution standard en deux étapes distinctes. Premièrement, la convolution spatiale est appliquée, où des filtres glissent sur les données d'entrée pour capturer des caractéristiques spatiales telles que les bords, les textures, et les motifs locaux. Cependant, au lieu d'appliquer un seul filtre à l'ensemble des canaux d'entrée, la deuxième étape, la convolution en profondeur, intervient. Elle consiste à appliquer des filtres distincts à chaque canal d'entrée, permettant au réseau de capturer des caractéristiques spécifiques à chaque couleur ou canal de l'image, comme la détection de teintes, de gradients, ou de textures uniques. L'avantage clé de la convolution en profondeur réside dans sa capacité à réduire le nombre de paramètres, optimisant ainsi l'efficacité computationnelle tout en préservant la richesse des informations extraites.

Fonction d'Activation et normalization

- La fonction d'activation ReLU (Rectified Linear Unit) est couramment utilisée après les opérations de convolution. La fonction ReLU introduit une non-linéarité en remplaçant toutes les valeurs négatives par zéro et en laissant les valeurs positives inchangées.
- La normalisation par lots (Batch Normalization) est une méthode intégrée pour stabiliser

le processus d'entraînement des réseaux neuronaux. Elle normalise les activations après une opération de convolution en ajustant la moyenne et l'écart-type des activations sur un mini-lot d'entraînement. Cette normalisation atténue le décalage covariant, améliorant la stabilité de l'entraînement en assurant des activations cohérentes. En agissant simultanément comme un régularisateur, Batch Normalization contribue à améliorer la généralisation du modèle, favorisant sa performance sur des données nouvelles ou non vues.

3.5 Segmentation à l'aide de l'Architecture Unet

La structure U-Net est un choix approprié en cas de données d'entraînement insuffisantes. Les U-Nets ont la capacité d'apprendre dans des environnements avec des quantités faibles à moyennes de données d'entraînement. Cette architecture a été conçue pour la première fois en 2015 par Ron-

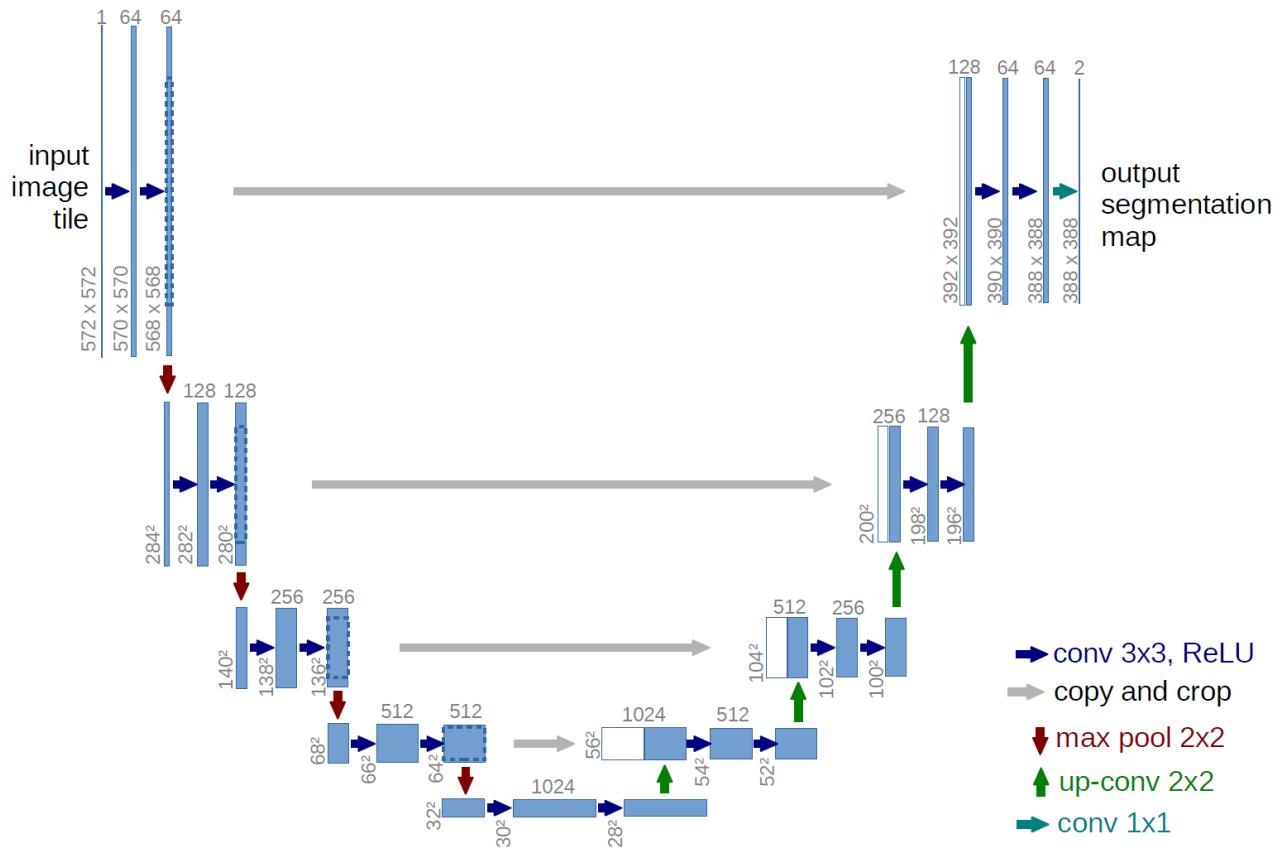


FIGURE 3.15: Architecture U-Net

neberger et al. pour les images biomédicales, et elle a été largement utilisée dans la segmentation d'images médicales depuis sa proposition. En raison de sa performance exceptionnelle, le U-Net et ses variantes ont été largement utilisés dans divers sous-domaines de la vision par ordinateur (CV).

Cette approche a été présentée lors de la conférence MICCAI 2015 et a été citée plus de 4000 fois. Jusqu'à présent, le U-Net a connu de nombreuses variantes.

La raison pour laquelle le U-Net est adapté à la segmentation d'images médicales est que sa structure peut combiner simultanément des informations de bas niveau et de haut niveau. Les informations de bas niveau aident à améliorer la précision, tandis que les informations de haut niveau aident à extraire des caractéristiques complexes.

L'architecture du réseau, qui comprend 23 couches, peut être divisée en trois parties comme indiqué dans la figure :

- *Le chemin de contraction ou de sous-échantillonnage (Encodeur).*
- *Le goulot d'étranglement horizontal.*
- *Le chemin d'expansion ou de suréchantillonnage (Décodeur).*

3.5.1 Le chemin de contraction ou de sous-échantillonnage (Encodeur) :

Cette partie de l'architecture U-Net est responsable de la capture des caractéristiques et contextes de l'image. Elle fonctionne par des couches successives de convolution et de pooling, réduisant progressivement la dimension spatiale tout en augmentant le nombre de canaux de caractéristiques. Chaque couche de convolution est généralement suivie d'une fonction d'activation non linéaire, comme ReLU, et le pooling est souvent effectué à l'aide de max-pooling. Ce processus de sous-échantillonnage aide à extraire et à condenser les informations importantes de l'image.

3.5.2 Le goulot d'étranglement horizontal :

Situé entre les chemins de contraction et d'expansion, ce goulot d'étranglement agit comme un pont dans l'architecture. Il traite les données les plus condensées et les plus abstraites de l'encodeur. Cette partie ne comporte généralement pas de couches de pooling et se compose de quelques couches de convolution. Le rôle principal de cette section est de traiter les caractéristiques les plus significatives qui ont été extraites et condensées par l'encodeur.

3.5.3 Le chemin d'expansion ou de suréchantillonnage (Décodeur) :

Le chemin d'expansion est le miroir du chemin de contraction. Il reconstruit l'image ou la carte de caractéristiques à partir des informations condensées par l'encodeur et le goulot d'étranglement. Ce processus implique des couches de convolution transposée (ou upconvolution) pour augmenter la dimension spatiale. Il intègre également des caractéristiques de l'encodeur grâce aux connexions dites "skip connections", qui aident à récupérer les informations spatiales perdues lors du processus de sous-échantillonnage. Le résultat est une sortie détaillée, souvent une carte de segmentation dans le cas des applications médicales.

3.6 Segmentation à l'aide de l'Architecture Unet++

UNet++ est Une Architecture de Segmentation Basée sur des Connexions de Saut Emboîtées et Denses , il se distingue de l'U-Net original de trois manières principales :

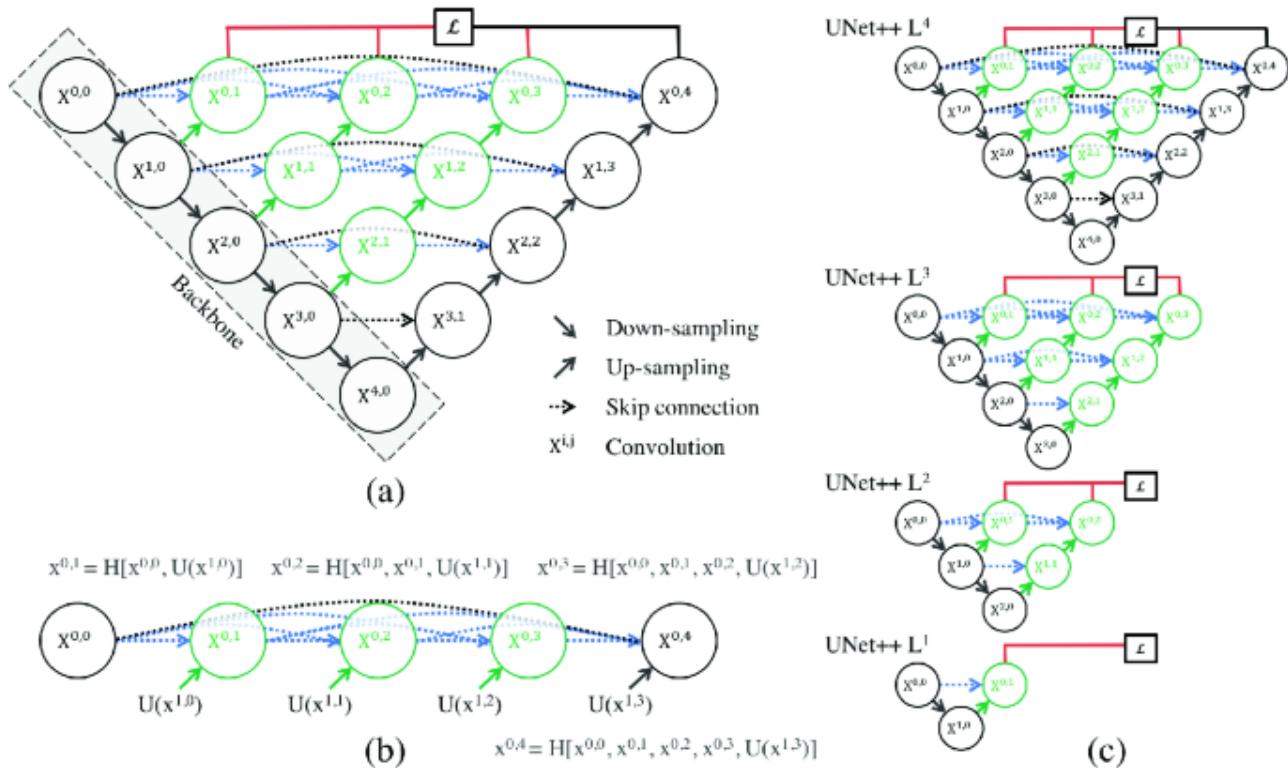


FIGURE 3.16: Architecture U-Net++

3.6.1 Couches de Convolution sur les Chemins de Saut :

UNet++ intègre des couches de convolution sur les chemins de saut (illustrés en vert). Ces couches servent à réduire l'écart sémantique entre les cartes de caractéristiques de l'encodeur et du décodeur. Cette intégration facilite une meilleure fusion des caractéristiques à différents niveaux de l'architecture.

3.6.2 Connexions de Saut Denses :

L'architecture utilise des connexions de saut denses (illustrées en bleu), améliorant ainsi le flux de gradient à travers le réseau. Ces connexions densifiées renforcent la propagation des informations et des gradients, ce qui contribue à une meilleure convergence du modèle lors de l'entraînement.

3.6.3 Supervision Profonde :

UNet++ est doté d'une supervision profonde (illustrée en rouge), permettant le pruning (élagage) du modèle et améliorant, ou dans le pire des cas, maintenant des performances comparables à celles obtenues avec une seule couche de perte. Cette supervision se fait à chaque niveau ($L1, L2, L3, L4$) en appliquant une fonction de perte intermédiaire qui compare la sortie du réseau à ce niveau avec la carte de segmentation cible correspondante.

3.6.3.1 Chemins de Saut Redessinés

Dans UNet++, le chemin de saut entre les nœuds $X0,0$ et $X1,3$ se compose d'un bloc de convolution dense avec trois couches de convolution. Chaque couche de convolution est précédée d'une couche de concaténation qui fusionne la sortie de la couche de convolution précédente du même bloc dense avec la sortie suréchantillonnée du bloc dense inférieur.

3.6.3.2 Supervision Profonde

À chaque niveau de l'architecture, une fonction de perte intermédiaire est appliquée, optimisant le modèle pour minimiser la somme de toutes ces pertes intermédiaires, en plus de la perte finale à la sortie de l'ensemble du réseau. Cela permet au réseau d'apprendre à la fois les caractéristiques de bas niveau et de haut niveau de manière plus efficace, améliorant ainsi les performances de segmentation.

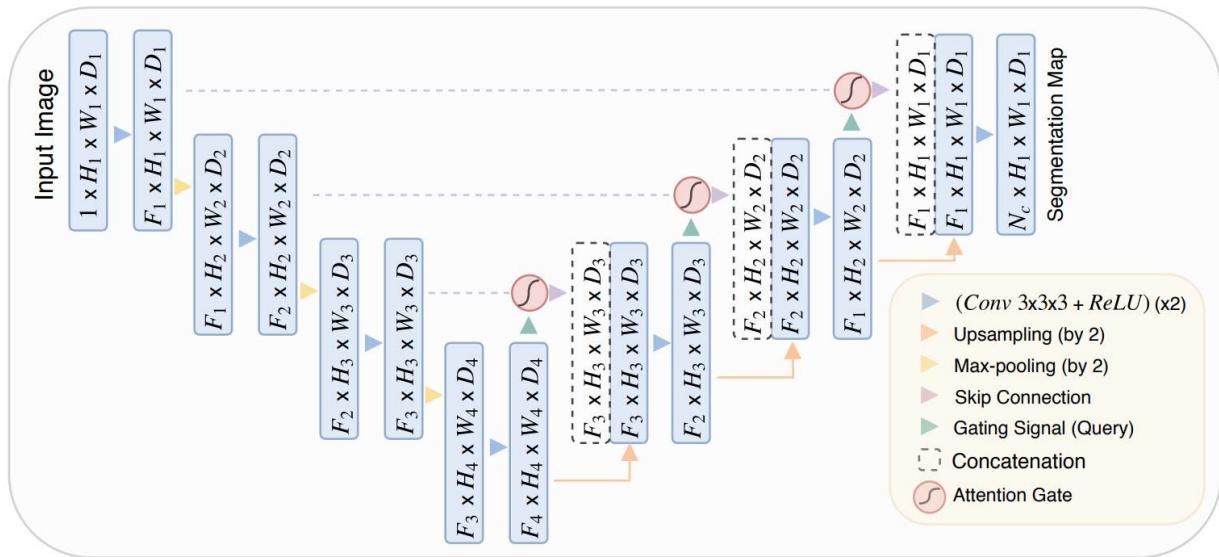
3.7 Segmentation à l'aide de l'Architecture Attention-Unet

L'Attention U-Net est une architecture de réseau neuronal convolutif (CNN) évoluée, dérivée de l'U-Net traditionnel, qui intègre des mécanismes d'attention spatiale à chaque étape de son processus.[14]

Cette architecture se décompose en plusieurs phases :

Encodeur avec Attention

L'encodeur constitue la première phase de l'architecture Attention U-Net, dédiée à l'extraction des caractéristiques significatives de l'image initiale. Dans le contexte de l'Attention U-Net, chaque couche de l'encodeur utilise des opérations de convolution pour identifier des motifs de bas niveau ,tels que des contours et des textures, et des opérations de pooling pour réduire la résolution spatiale. La structure de l'U-Net avec attention intègre un mécanisme d'attention à chaque étape de l'encodeur. Ce mécanisme agit en pondérant les activations mettant en avant les zones jugées essentielles pour



la détection des tumeurs cérébrales. Il permet au réseau de se concentrer de manière sélective sur des caractéristiques spécifiques tout en réduisant l'impact du bruit ou des zones moins pertinentes, ce qui conduit à une amélioration de la qualité de la représentation apprise.

Connexions Résiduelles avec Attention

Ces connexions établissent un lien direct entre les couches correspondantes de l'encodeur et du décodeur, facilitant ainsi le transfert efficace d'informations cruciales. Dans cette variante de l'U-Net, des mécanismes d'attention sont incorporés dans ces connexions résiduelles. Ces mécanismes permettent d'ajuster l'importance des caractéristiques avant de les ajouter. Cette démarche garantit que seules les informations jugées cruciales soient transmises au décodeur. En préservant de manière sélective les détails spatiaux essentiels, même après des étapes successives de pooling et de déconvolution, cette approche enrichit significativement la qualité des informations transmises pour la détection de tumeurs.

Décodeur avec Attention

Le décodeur est la partie de l'U-Net qui restaure la résolution spatiale de l'image. À chaque étape du décodeur, avant l'opération d'upsampling, un bloc d'attention est introduit. Ce bloc d'attention agit comme un guide, aidant le décodeur à se concentrer sur les caractéristiques les plus importantes pour reconstruire l'image segmentée. L'attention est particulièrement bénéfique lors de l'upsampling, où le modèle doit prendre des décisions sur la reconstruction des détails spatiaux à partir de caractéristiques agrégées.

Couche de Sortie

La couche de sortie produit la carte de segmentation, indiquant la probabilité que chaque pixel appartienne à une classe particulière. Dans l'U-Net avec attention, cette couche de sortie est guidée par

l'ensemble des mécanismes d'attention présents tout au long de l'architecture. Par conséquent, la carte de segmentation finale est modulée par les caractéristiques importantes mises en évidence par les mécanismes d'attention à chaque étape du réseau.

Évaluation Comparative des Modèles de Segmentation

Préambule

Ce chapitre se concentre sur la comparaison approfondie des modèles de segmentation d'images et leur implémentation. L'objectif est de fournir un aperçu clair des forces, des limitations et des applications de chaque modèle dans la segmentation pour la Détection des Tumeurs Cérébrales.

4.1 Présentation des outils utilisés

4.1.1 Environnement d'exécution

La plateforme Kaggle a été le pilier de notre environnement d'exécution pour ce projet. Grâce à ses ressources en cloud, notamment des GPU puissants, et ses fonctionnalités d'apprentissage automatique intégrées, Kaggle a joué un rôle pivot dans l'ensemble du processus, de la conception à l'évaluation des modèles de segmentation.



4.1.2 Bibliothèques et frameworks

L'utilisation de bibliothèques et de frameworks tels que TensorFlow et Keras a été essentielle dans le développement de nos modèles. TensorFlow a été notre principal outil pour créer et former ces

modèles, tandis que Keras nous a offert une flexibilité et une rapidité de prototypage précieuses.



4.1.3 Langage de programmation

Python a été le langage de programmation central pour ce projet de segmentation. Sa simplicité, sa richesse en bibliothèques d'apprentissage automatique et sa polyvalence en font un choix naturel pour la mise en œuvre de modèles complexes comme ceux de segmentation d'images.



4.2 Base de données

4.2.1 Description détaillée de la base de données

Un groupe de données comprenant des images IRM du cerveau a été utilisé pour cette étude. Les images, obtenues par IRM (Imagerie par Résonance Magnétique), mettent en évidence des parties du cerveau présentant des anomalies ou des différences. Ces images sont accompagnées de masques manuellement créés pour identifier les zones anormales dans des séquences spécifiques appelées FLAIR (Fluid Attenuated Inversion Recovery), une méthode particulière d'obtention d'images cérébrales.[3]

Le FLAIR est une séquence IRM spécifique utilisée pour atténuer le signal du liquide cérébrospinal, permettant une meilleure visualisation de certaines structures cérébrales. Il est principalement utilisé pour mettre en évidence des tissus ou des lésions spécifiques en atténuant le signal du liquide entourant le cerveau.[7]

La base de données TCGA (The Cancer Genome Atlas) contient des informations génomiques de patients atteints de divers cancers, tandis que TCIA (The Cancer Imaging Archive) se concentre sur les images médicales, comme les IRM, provenant de patients cancéreux.

Les gliomes de grade inférieur, des tumeurs cérébrales se développant plus lentement, ont été étudiés dans cette recherche.

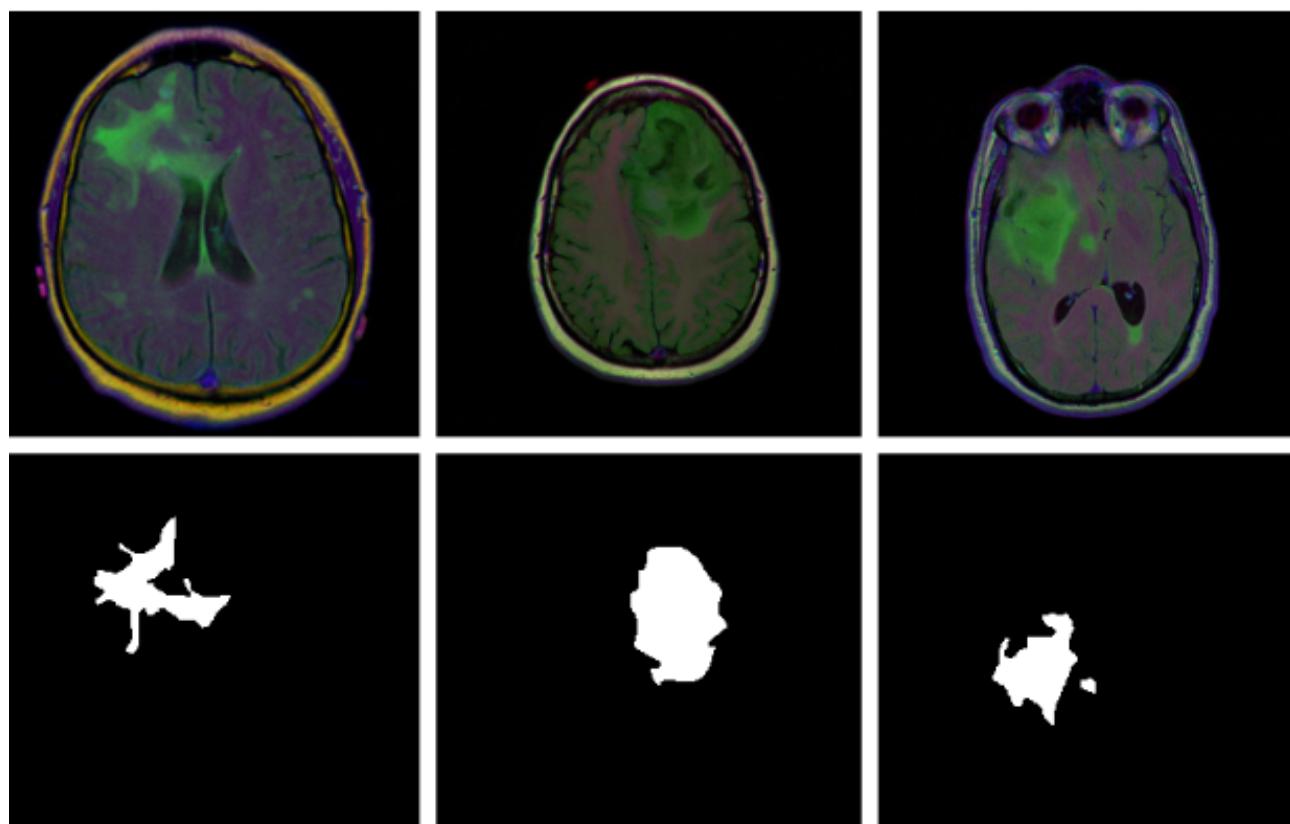


FIGURE 4.1: Images cérébrales segmentés manuellement

4.2.2 Composition de l'échantillon

L'échantillon de 110 patients provenait de cinq institutions différentes, chaque patient ayant subi différentes séquences d'imagerie cérébrale.

- 101 patients avaient toutes les séquences d'images disponibles.
- 9 patients manquaient de certaines séquences après l'injection d'un produit de contraste.
- 6 patients n'avaient pas toutes les séquences avant l'injection du produit de contraste.

Dans cette étude, les images FLAIR ont été utilisées comme substitut lorsque d'autres types d'images n'étaient pas disponibles.

4.2.3 Annotations et divisions

Les images FLAIR ont été annotées manuellement pour former des données d'entraînement pour l'algorithme de segmentation automatique. Le groupe final de 110 patients provenait des cinq institutions suivantes : l'Université Thomas Jefferson (TCGA-CS, 16 patients), l'hôpital Henry Ford (TCGA-DU, 45 patients), UNC (TCGA-EZ, 1 patient), Case Western (TCGA-FG, 14 patients), Case Western - St. Joseph's (TCGA-HT, 34 patients) de la collection TCGA LGG.

4.2.4 Caractéristiques génomiques

```
data = pd.read_csv('../input/lgg-mri-segmentation/kaggle_3m/data.csv')
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 110 entries, 0 to 109
Data columns (total 18 columns):
 #   Column           Non-Null Count  Dtype  
 ---  --  
 0   Patient          110 non-null    object  
 1   RNASEqCluster    92 non-null     float64 
 2   MethylationCluster 109 non-null    float64 
 3   miRNACluster     110 non-null    int64  
 4   CNCluster         108 non-null    float64 
 5   RPPACluster       98 non-null     float64 
 6   OncosignCluster   105 non-null    float64 
 7   COCCluster        110 non-null    int64  
 8   histological_type 109 non-null    float64 
 9   neoplasm_histologic_grade 109 non-null    float64 
 10  tumor_tissue_site 109 non-null    float64 
 11  laterality        109 non-null    float64 
 12  tumor_location    109 non-null    float64 
 13  gender            109 non-null    float64
```

Les données génomiques analysées comprenaient la méthylation de l'ADN, l'expression génique, le nombre de copies de l'ADN, l'expression de microARN et une mutation spécifique appelée co-

délétion 1p/19q liée à l'IDH. Ces caractéristiques ont été utilisées pour classifier les gliomes de grade inférieur en différents sous-types moléculaires.

4.2.5 Prétraitement des données

Dans notre démarche, chaque lot de données généré comprend un ensemble d'images originales accompagnées de masques de segmentation correspondants. Les étapes clés du prétraitement incluent le redimensionnement uniforme des images et des masques à une résolution de 256x256 pixels. Avec $img_h = 256$ et $img_w = 256$.

```
#resizing and converting them to array of type float64
    img = cv2.resize(img, (self.img_h, self.img_w))
    img = np.array(img, dtype = np.float64)

    mask = cv2.resize(mask, (self.img_h, self.img_w))
    mask = np.array(mask, dtype = np.float64)
```

Suivi d'une normalisation des valeurs de pixels spécifique à chaque image. Cette normalisation vise à standardiser les valeurs des pixels en les ajustant par la soustraction de la moyenne et la division par l'écart type.

```
#standardising
    img -= img.mean()
    img /= img.std()
    if np.sum(mask) > 0:
        mask -= mask.mean()
        mask /= mask.std()
```

4.3 Les métrics de performance

Dans le cadre de notre étude comparative des approches de détection et de segmentation des tumeurs cérébrales, le choix des critères de performance revêt une importance cruciale pour évaluer de manière exhaustive les résultats obtenus par chaque modèle. Ces critères ont été sélectionnés pour

offrir une évaluation complète de la performance des modèles, prenant en compte à la fois la précision de la segmentation et la capacité à détecter correctement les régions de tumeurs dans les images cérébrales.

4.3.1 Indice de Similarité Jaccard (IoU)

L'Indice de Similarité Jaccard (IoU), également connu sous le nom de Coefficient de Jaccard, est une métrique d'évaluation utilisée fréquemment dans les tâches de segmentation d'images, y compris la segmentation sémantique et la détection d'objets. L'IoU mesure le degré de chevauchement entre la prédiction d'un modèle et la vérité terrain. Il est défini comme le rapport entre l'intersection de l'aire prédite et de l'aire réelle sur l'union de ces deux aires. La formule mathématique de l'IoU est donnée par :

$$IoU = \frac{|A \cap B|}{|A \cup B|}$$

Mais pour la classification binaire, cela s'écrit :

$$IoU = \frac{TP}{TP + FN + FP}$$

où :

- *TP* représente les Vrais Positifs
- *FN* représente les Faux Négatifs
- *FP* représente les Faux Positifs



FIGURE 4.2: Intersection Over Union (IoU)

Cependant, graphiquement, la formule de l'indice de Jaccard est généralement illustrée de la manière suivante :

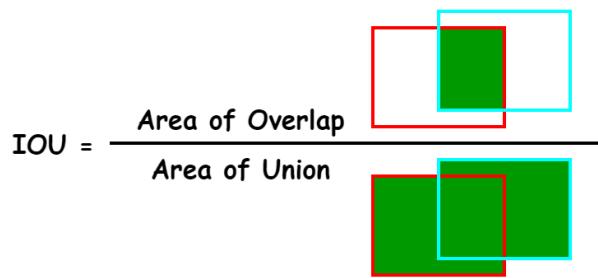


FIGURE 4.3: Représentation graphique d'IoU

Le score IoU sera élevé s'il y a beaucoup de chevauchement entre les boîtes de vérité anticipée et de terrain. En revanche, un faible chevauchement entraînera un faible score IoU. Un score IoU de 1 indique une correspondance parfaite entre la boîte projetée et la boîte de vérité terrain, tandis qu'un score de 0 signifie qu'il n'y a pas de chevauchement entre les boîtes.

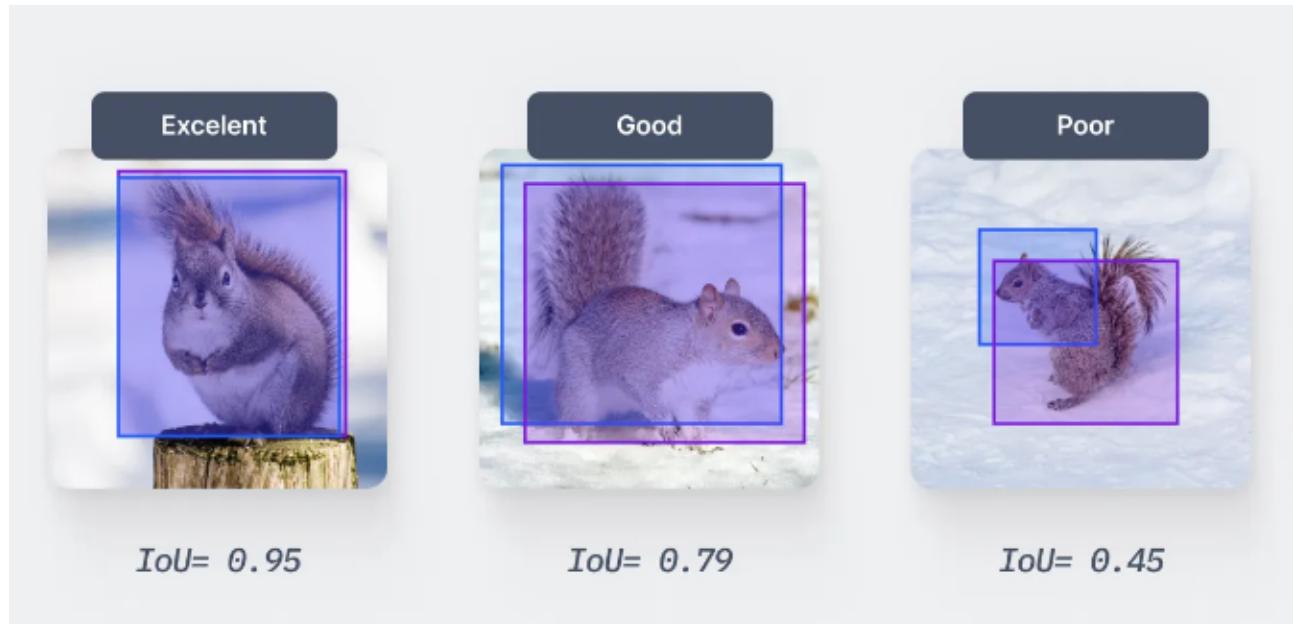


FIGURE 4.4: Performance comparative de l'indice de Jaccard (IoU)

4.3.2 Coefficient Dice (Dice Coef)

Le Coefficient Dice, également connu sous le nom de F1-score binaire, est une métrique d'évaluation couramment utilisée pour mesurer la similarité entre deux ensembles. Il est souvent employé dans le contexte de la segmentation d'images, où il évalue la concordance entre la prédiction d'un modèle et la vérité terrain. Mathématiquement, le Coefficient Dice est défini comme le double de la proportion de l'intersection entre deux ensembles divisée par la somme de leurs aires. La formule

s'exprime de la manière suivante :

$$DiceCoef = \frac{2 \times TP}{2 \times TP + FP + FN}$$

où :

- TP représente les Vrais Positifs
- FN représente les Faux Négatifs
- FP représente les Faux Positifs

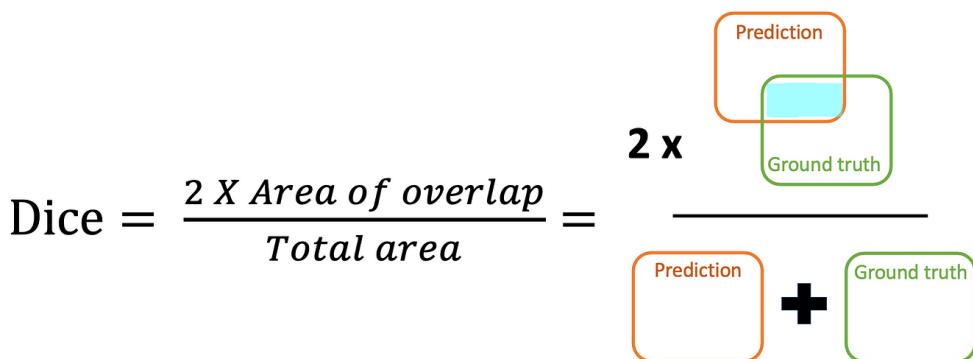


FIGURE 4.5: Représentation graphique de Dice coefficient

Le Coefficient Dice varie de 0 à 1, où 0 indique aucune similarité (pas de chevauchement) et 1 indique une similarité parfaite (chevauchement complet). Plus le Coefficient Dice est élevé, meilleure est la concordance entre la prédiction et la vérité terrain. Cette métrique est particulièrement adaptée pour évaluer la qualité des contours et des formes dans les tâches de segmentation, car elle pénalise les fausses prédictions et valorise un chevauchement précis entre les zones prédites et réelles.

4.3.3 Précision (Precision)

La précision, dans le contexte de l'évaluation des performances d'un modèle, est une mesure qui quantifie le nombre de prédictions correctes par rapport au nombre total de prédictions effectuées par le modèle. Cette métrique met l'accent sur la capacité du modèle à minimiser les faux positifs et sa formule s'exprime de la manière suivante :

$$Précision = \frac{TP}{TP + FP}$$

En d'autres termes, la précision mesure la proportion de prédictions positives effectuées par le modèle qui étaient effectivement correctes. Elle est souvent utilisée en conjonction avec le rappel pour évaluer de manière globale la performance d'un modèle.

4.3.4 Rappel (Recall)

Le rappel (recall), également appelé sensibilité ou taux de vrais positifs, est une mesure d'évaluation utilisée pour évaluer la capacité d'un modèle à identifier correctement toutes les occurrences d'une classe positive. Le rappel est particulièrement important dans les tâches où la détection des vrais positifs est cruciale, même au détriment de certains faux positifs. La formule du rappel est définie comme le ratio du nombre de vrais positifs (VP) sur la somme des vrais positifs et des faux négatifs (FN) :

$$\text{Rappel} = \frac{TP}{TP + FN}$$

Le rappel est particulièrement utile dans les situations où la non-détection de vrais positifs peut avoir des conséquences graves, par exemple dans les applications médicales telles que la détection de maladies. Lorsque le rappel est élevé, cela signifie que le modèle réussit à identifier la plupart, voire toutes, les instances de la classe positive. Cependant, un rappel élevé peut parfois être accompagné d'un nombre plus élevé de faux positifs, ce qui peut être un compromis à considérer en fonction du domaine d'application.

4.4 La fonction du coût

4.4.1 Dice Loss

Dice Loss est une fonction de perte fondamentale utilisée lors de l'entraînement des modèles de segmentation. Son rôle principal est de mesurer la dissimilitude entre deux ensembles de données, généralement les masques prédits par le modèle et les masques de vérité terrain. Cela permet d'ajuster les paramètres du modèle pour réduire cette dissimilitude et améliorer la précision de la segmentation. Cette fonction est basée sur le Coefficient Dice et sa formule mathématique s'exprime comme suit :

$$\text{Dice Loss} = 1 - \frac{2 \times TP}{2 \times TP + FP + FN}$$

4.4.2 L'objectif

L'objectif principal lors de l'utilisation du Dice Loss est de minimiser cette fonction pendant l'entraînement du modèle. En minimisant le Dice Loss, on cherche à maximiser le Coefficient Dice, une mesure de similarité entre les masques prédits et les masques de référence.

Le Dice Loss est particulièrement couramment employé dans les tâches de segmentation d'images, notamment dans les domaines médicaux. Ce choix est motivé par sa capacité à pénaliser de manière plus significative les erreurs de segmentation, par rapport à d'autres fonctions de perte comme la perte d'entropie croisée (cross-entropy loss). De plus, il est adapté aux situations où les classes à segmenter présentent des déséquilibres significatifs, rendant la segmentation plus complexe.

Cette fonction de perte s'avère être un outil essentiel dans l'entraînement des modèles de segmentation, offrant une meilleure gestion des déséquilibres entre les classes et une amélioration de la précision des résultats de segmentation.

4.5 Les optimiseurs

Pour l'entraînement de nos modèles, nous avons utilisé une approche basée sur des algorithmes d'optimisation avancés afin de minimiser la fonction de coût et d'améliorer les performances du réseau. Deux optimiseurs ont été explorés au cours de l'expérimentation, à savoir Adam et SGD.

Optimiseur Adam

L'optimiseur Adam, ou Adaptive Moment Estimation, a été choisi en raison de sa capacité à combiner les avantages de la méthode du gradient stochastique et de l'algorithme d'estimation adaptative des moments. Adam s'est avéré efficace pour converger rapidement vers des minima locaux tout en adaptant dynamiquement le taux d'apprentissage pour chaque paramètre du modèle.

Paramètres Importants

- α (alpha) : Taux d'apprentissage initial.
- β_1 (beta_1) : Facteur de décroissance du premier moment (moyenne mobile des gradients).
- β_2 (beta_2) : Facteur de décroissance du deuxième moment (moyenne mobile des carrés des gradients).
- ϵ (epsilon) : Terme ajouté au dénominateur pour éviter la division par zéro.

Optimiseur SGD

L'optimiseur SGD, ou Stochastic Gradient Descent a également été considéré dans notre approche

d'entraînement. L'optimiseur SGD est basé sur le principe de la descente de gradient. Il ajuste les poids du modèle en fonction du gradient de la fonction de coût par rapport aux poids. Cela permet de minimiser la fonction de coût et d'optimiser les performances du modèle. La caractéristique "stochastique" signifie que les mises à jour des poids ne sont pas basées sur l'ensemble complet des données d'entraînement, mais sur un sous-ensemble (un seul exemple ou un mini-lot). Cela rend l'optimisation plus rapide, surtout dans le cas de grands ensembles de données. Certains variantes de SGD incluent un terme de momentum qui introduit une composante d'inertie. Cela aide à accélérer l'optimisation et à éviter que le modèle ne reste coincé dans des minima locaux.

Taux d'Apprentissage

Le taux d'apprentissage (α) est un paramètre clé dans l'algorithme SGD et il influence la taille des pas que l'optimiseur prend lors de la mise à jour des poids du modèle. En d'autres termes, il détermine la vitesse à laquelle le modèle apprend et converge vers un minimum de la fonction de coût.

4.6 Implémentation des Méthodes

4.6.1 DeepLabV3+

DeepLabV3+ se distingue par ses caractéristiques distinctives, notamment sa capacité à capturer des détails fins et à maintenir la précision de la segmentation, même pour des objets de petite taille dans les images. Son architecture repose sur une combinaison novatrice de plusieurs éléments clés, tels que l'utilisation d'atrous convolutions pour augmenter le champ réceptif sans augmenter le nombre de paramètres, et l'intégration de modules de mise à l'échelle spatiale pour capturer des informations à différentes échelles dans une image. Cette approche permet à DeepLab de réaliser des segmentations sémantiques de haute qualité, en attribuant des étiquettes de classe à chaque pixel d'une image, offrant ainsi une compréhension fine et précise du contenu visuel. DeepLab est couramment utilisé dans divers domaines, tel que la segmentation sémantique des images médicales.

Voici le modèle implémenté :

```

from tensorflow.keras.layers import Conv2D, BatchNormalization, Activation, MaxPool2D, Conv2DTranspose, Concatenate, Input
from tensorflow.keras.layers import AveragePooling2D, GlobalAveragePooling2D, UpSampling2D, Reshape, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.applications import ResNet50

def SqueezeAndExcite(inputs, ratio=8):
    # Fonction Squeeze and Excite
    init = inputs
    filters = init.shape[-1]
    se_shape = (1, 1, filters)

    se = GlobalAveragePooling2D()(init)
    se = Reshape(se_shape)(se)
    se = Dense(filters // ratio, activation='relu', kernel_initializer='he_normal', use_bias=False)(se)
    se = Dense(filters, activation='sigmoid', kernel_initializer='he_normal', use_bias=False)(se)
    x = init * se
    return x

def ASPP(inputs):

```

```

# Fonction ASPP
shape = inputs.shape
y1 = AveragePooling2D(pool_size=(shape[1], shape[2]))(inputs)
y1 = Conv2D(256, 1, padding="same", use_bias=False)(y1)
y1 = BatchNormalization()(y1)
y1 = Activation("relu")(y1)
y1 = UpSampling2D((shape[1], shape[2]), interpolation="bilinear")(y1)

y2 = Conv2D(256, 1, padding="same", use_bias=False)(inputs)
y2 = BatchNormalization()(y2)
y2 = Activation("relu")(y2)

y3 = Conv2D(256, 3, padding="same", use_bias=False, dilation_rate=6)(inputs)
y3 = BatchNormalization()(y3)
y3 = Activation("relu")(y3)

y4 = Conv2D(256, 3, padding="same", use_bias=False, dilation_rate=12)(inputs)
y4 = BatchNormalization()(y4)
y4 = Activation("relu")(y4)

y5 = Conv2D(256, 3, padding="same", use_bias=False, dilation_rate=18)(inputs)
y5 = BatchNormalization()(y5)
y5 = Activation("relu")(y5)

```

```

y = Concatenate()([y1, y2, y3, y4, y5])
y = Conv2D(256, 1, padding="same", use_bias=False)(y)
y = BatchNormalization()(y)
y = Activation("relu")(y)

return y

def deeplabv3_plus(shape):
    # Définition du modèle DeepLabV3+
    inputs = Input(shape)
    encoder = ResNet50(weights="/kaggle/input/resnet50-weights/resnet50_weights_tf_dim_orderi
ng_tf_kernels_notop.h5",
                        include_top=False, input_tensor=inputs)

    image_features = encoder.get_layer("conv4_block6_out").output
    x_a = ASPP(image_features)
    x_a = UpSampling2D((4, 4), interpolation="bilinear")(x_a)

    x_b = encoder.get_layer("conv2_block2_out").output
    x_b = Conv2D(filters=48, kernel_size=1, padding='same', use_bias=False)(x_b)
    x_b = BatchNormalization()(x_b)
    x_b = Activation('relu')(x_b)

    x = Concatenate()([x_a, x_b])
    x = SqueezeAndExcite(x)

    x = Conv2D(filters=256, kernel_size=3, padding='same', use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)

    x = Conv2D(filters=256, kernel_size=3, padding='same', use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = SqueezeAndExcite(x)

    x = UpSampling2D((4, 4), interpolation="bilinear")(x)
    x = Conv2D(1, 1)(x)
    x = Activation("sigmoid")(x)

    model = Model(inputs, x)
    return model

if __name__ == "__main__":
    model = deeplabv3_plus((256, 256, 3))
    model.summary()

```

```
=====
=====
Total params: 17869697 (68.17 MB)
Trainable params: 17834913 (68.03 MB)
Non-trainable params: 34784 (135.88 KB)
-----
```

```
from tensorflow.keras.callbacks import ModelCheckpoint, CSVLogger, ReduceLROnPlateau, EarlyStopping, TensorBoard
```

```
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.metrics import Recall, Precision
```

```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-5),
              loss=dice_loss,
              metrics=[dice_coef, iou, Recall(), Precision()])
```

4.6.1.1 L'entraînement du modèle

Pendant la phase d'entraînement de DeepLabV3+, plusieurs étapes critiques sont utilisées pour améliorer les performances du modèle et garantir un apprentissage optimal. Les rappels (callbacks) jouent un rôle central en surveillant la progression de l'entraînement et en ajustant dynamiquement les paramètres. En utilisant les rappels de Keras de TensorFlow, des méthodes telles que EarlyStopping, ReduceLROnPlateau et ModelCheckpoint sont intégrées dans le processus d'entraînement. ReduceLROnPlateau diminue efficacement le taux d'apprentissage lorsque l'amélioration du modèle stagne, permettant des ajustements plus fins pour faciliter la convergence. Pendant ce temps, ModelCheckpoint sauvegarde régulièrement les poids du modèle, garantissant que seule la version la plus performante est préservée. Ces rappels contribuent collectivement à un processus d'entraînement plus robuste et adaptatif, permettant à DeepLabV3+ d'apprendre des motifs complexes et de produire des résultats de segmentation précis pour les images par résonance magnétique du cerveau. De plus, la procédure d'entraînement implique de définir des tailles de lots appropriées, des époques et des étapes de validation pour optimiser efficacement les performances du modèle sur de multiples itérations.

```
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint

callbacks = [
    ReduceLROnPlateau(factor=0.1, min_lr=1e-5, verbose=1),
    ModelCheckpoint('model-brain-mri-6.h5', verbose=1, save_best_only=True, save_weights_only
=True)
]
```

```
STEP_SIZE_TRAIN = timage_generator.n/BATCH_SIZE
STEP_SIZE_VALID = vimage_generator.n/BATCH_SIZE
```

```
results = model.fit(train_gen,
                     steps_per_epoch=STEP_SIZE_TRAIN,
                     batch_size=BATCH_SIZE,
                     epochs=100,
                     callbacks=callbacks,
                     validation_data=valid_gen,
                     validation_steps=STEP_SIZE_VALID)
```

4.6.1.2 Les résultats de l'entraînement

```
Epoch 1/100
79/78 [=====] - ETA: 0s - loss: 0.9637 - dice_c
oef: 0.0363 - iou: 0.0185 - recall: 0.9996 - precision: 0.0200
Epoch 1: val_loss improved from inf to 0.97835, saving model to model-br
ain-mri-6.h5
78/78 [=====] - 191s 1s/step - loss: 0.9637 - d
ice_coef: 0.0363 - iou: 0.0185 - recall: 0.9996 - precision: 0.0200 - va
l_loss: 0.9783 - val_dice_coef: 0.0215 - val_iou: 0.0109 - val_recall:
0.9945 - val_precision: 0.0121 - lr: 1.0000e-05
Epoch 2/100
79/78 [=====] - ETA: 0s - loss: 0.9307 - dice_c
oef: 0.0693 - iou: 0.0361 - recall: 0.9918 - precision: 0.0616
Epoch 2: val_loss improved from 0.97835 to 0.97691, saving model to mode
l-brain-mri-6.h5
78/78 [=====] - 80s 1s/step - loss: 0.9307 - di
ce_coef: 0.0693 - iou: 0.0361 - recall: 0.9918 - precision: 0.0616 - val
_loss: 0.9769 - val_dice_coef: 0.0234 - val_iou: 0.0118 - val_recall: 0.
5501 - val_precision: 0.0490 - lr: 1.0000e-05

Epoch 99/100
79/78 [=====] - ETA: 0s - loss: 0.0887 - dice_c
oef: 0.9113 - iou: 0.8382 - recall: 0.9008 - precision: 0.9564
Epoch 99: val_loss improved from 0.12289 to 0.12250, saving model to mod
el-brain-mri-6.h5
78/78 [=====] - 80s 1s/step - loss: 0.0887 - di
ce_coef: 0.9113 - iou: 0.8382 - recall: 0.9008 - precision: 0.9564 - val
_loss: 0.1225 - val_dice_coef: 0.8779 - val_iou: 0.7841 - val_recall: 0.
8954 - val_precision: 0.8972 - lr: 1.0000e-05
Epoch 100/100
79/78 [=====] - ETA: 0s - loss: 0.0828 - dice_c
oef: 0.9171 - iou: 0.8472 - recall: 0.9027 - precision: 0.9601
Epoch 100: val_loss improved from 0.12250 to 0.12248, saving model to mo
del-brain-mri-6.h5
78/78 [=====] - 80s 1s/step - loss: 0.0828 - di
ce_coef: 0.9171 - iou: 0.8472 - recall: 0.9027 - precision: 0.9601 - val
_loss: 0.1225 - val_dice_coef: 0.8777 - val_iou: 0.7825 - val_recall: 0.
8963 - val_precision: 0.8904 - lr: 1.0000e-05
```

Les résultats de l'entraînement sur 100 époques indiquent une évolution significative des performances du modèle DeepLabV3+. Initialement, la perte (loss) était de 0.9637 avec un coefficient Dice de 0.0363, une intersection sur union (IoU) de 0.0185, une sensibilité (recall) de 0.9996 et une précision (precision) de 0.0200. Cependant, au fil des époques, ces métriques se sont considérablement améliorées. À l'époque 100, la perte est descendue à 0.0828, avec un coefficient Dice de 0.9171, une IoU de 0.8472, une sensibilité de 0.9027 et une précision de 0.9601. Ce progrès remarquable est également reflété dans les performances sur les données de validation, où la perte est passée de 0.97835 à 0.12248 et le coefficient Dice de 0.0215 à 0.8777 sur la même période. Ces résultats indiquent une convergence réussie du modèle vers des performances bien meilleures, démontrant sa capacité à apprendre et à segmenter avec précision les images par résonance magnétique du cerveau.

Voici ce graphique illustrant ces métriques de performance ainsi que la fonction de coût (Dice Loss) tout au long de 100 époques :

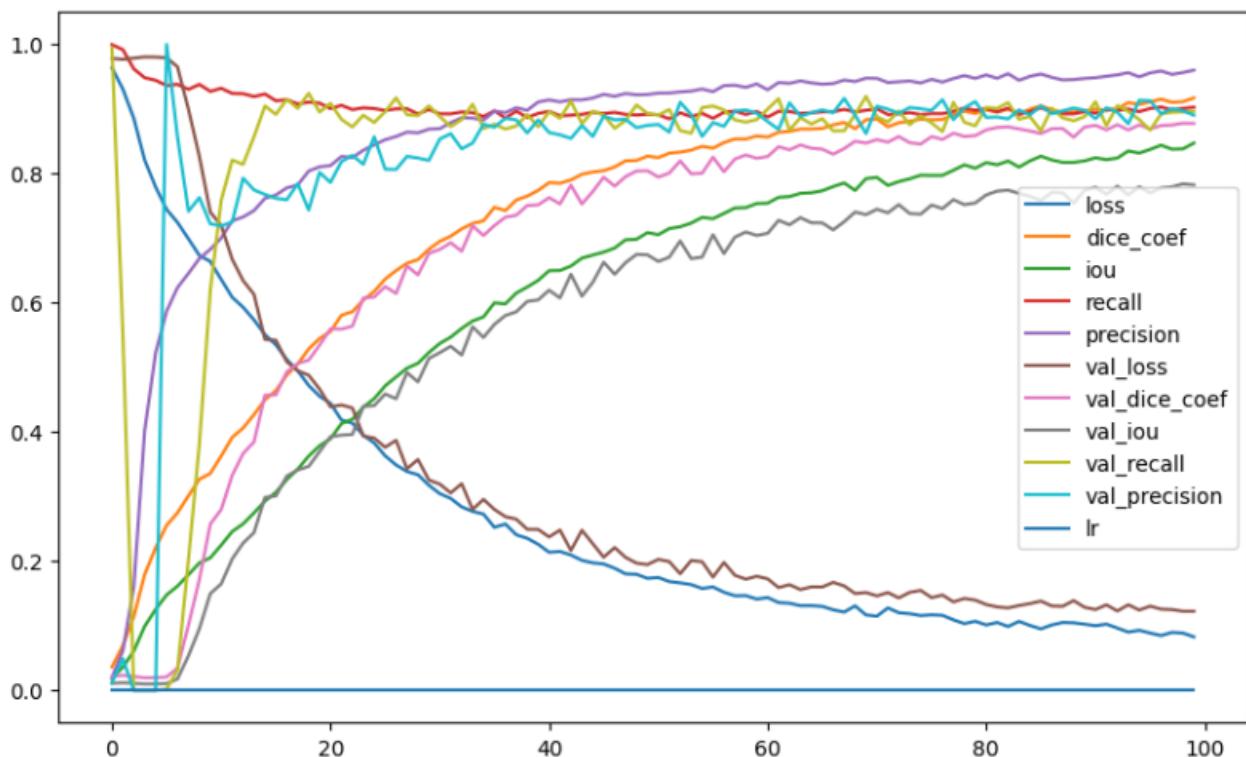


FIGURE 4.6: Synthèse de la Performance du Modèle DeepLabV3+ : Données d'Entraînement et de Validation

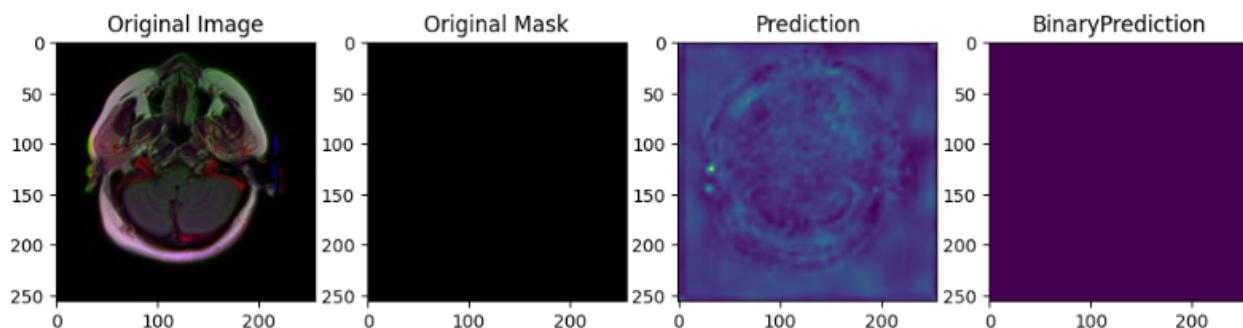
4.6.1.3 L'évaluation du modèle

Les résultats de l'évaluation du modèle sur les données de test montrent une perte basse, un coefficient Dice élevé, ainsi qu'un bon équilibre entre rappel (recall) et précision. Cela suggère que le modèle parvient à bien généraliser sur des données qu'il n'a pas vues pendant l'entraînement, offrant ainsi des performances prometteuses.

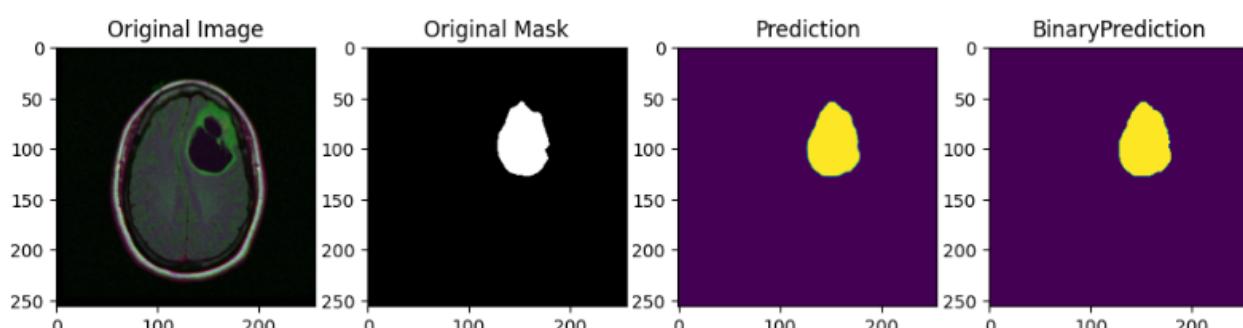
```
eval_results = model.evaluate(test_gen, steps=STEP_SIZE_VALID, verbose=1)
```

```
19/19 [=====] - 9s 463ms/step - loss: 0.1176 - dice_coef: 0.8824  
- iou: 0.7907 - recall: 0.9066 - precision: 0.8982
```

4.6.1.4 Tester le modèle sur les images du test



```
1/1 [=====] - 0s 42ms/step
```



4.6.2 PSPNet

Le PSPNet, abréviation de Pyramid Scene Parsing Network, est reconnu pour son architecture novatrice qui vise à capturer des informations contextuelles à différentes échelles spatiales. Cette

méthode s'appuie sur des pyramides de contexte de différentes tailles pour enrichir la compréhension de la scène visuelle. L'un de ses aspects distinctifs est l'utilisation de blocs de pyramides de pooling afin de saisir des détails contextuels à différentes résolutions. Ces blocs permettent une analyse à plusieurs échelles, améliorant ainsi la capacité du modèle à interpréter des objets de différentes tailles dans une image. Cette approche raffinée de capture de contexte permet au PSPNet de réaliser des segmentations de haute qualité, en attribuant des étiquettes sémantiques aux pixels, offrant ainsi une segmentation précise même pour des images complexes et riches en détails. Le PSPNet est largement utilisé dans des applications de vision par ordinateur, notamment dans la segmentation sémantique d'images médicales et de scènes extérieures.

Voici le modèle implémenté :

```
def conv_block(X,filters,block):
    # residual block with dilated convolutions
    # add skip connection at last after doing convolution operation to input X

    b = 'block_'+str(block) + '_'
    f1,f2,f3 = filters
    X_skip = X
    # block_a
    X = Convolution2D(filters=f1,kernel_size=(1,1),dilation_rate=(1,1),
                       padding='same',kernel_initializer='he_normal',name=b+'a')(X)
    X = BatchNormalization(name=b+'batch_norm_a')(X)
    X = LeakyReLU(alpha=0.2,name=b+'leakyrelu_a')(X)
    # block_b
    X = Convolution2D(filters=f2,kernel_size=(3,3),dilation_rate=(2,2),
                       padding='same',kernel_initializer='he_normal',name=b+'b')(X)
    X = BatchNormalization(name=b+'batch_norm_b')(X)
    X = LeakyReLU(alpha=0.2,name=b+'leakyrelu_b')(X)
    # block_c
    X = Convolution2D(filters=f3,kernel_size=(1,1),dilation_rate=(1,1),
                       padding='same',kernel_initializer='he_normal',name=b+'c')(X)
    X = BatchNormalization(name=b+'batch_norm_c')(X)
    # skip_conv
```

```

X_skip = Convolution2D(filters=f3,kernel_size=(3,3),padding='same',name=b+'skip_conv')(X)
X_skip = BatchNormalization(name=b+'batch_norm_skip_conv')(X_skip)
# block_c + skip_conv
X = Add(name=b+'add')([X,X_skip])
X = ReLU(name=b+'relu')(X)
return X

def base_feature_maps(input_layer):
    # base convolution module to get input image feature maps

    # block_1
    base = conv_block(input_layer,[32,32,64],'1')
    # block_2
    base = conv_block(base,[64,64,128],'2')
    # block_3
    base = conv_block(base,[128,128,256],'3')
    return base

def pyramid_feature_maps(input_layer):
    # pyramid pooling module

    base = base_feature_maps(input_layer)
    # red

```

```

# red
red = GlobalAveragePooling2D(name='red_pool')(base)
red = tf.keras.layers.Reshape((1,1,256))(red)
red = Convolution2D(filters=64,kernel_size=(1,1),name='red_1_by_1')(red)
red = UpSampling2D(size=256,interpolation='bilinear',name='red_upsampling')(red)
# yellow
yellow = AveragePooling2D(pool_size=(2,2),name='yellow_pool')(base)
yellow = Convolution2D(filters=64,kernel_size=(1,1),name='yellow_1_by_1')(yellow)
yellow = UpSampling2D(size=2,interpolation='bilinear',name='yellow_upsampling')(yellow)
# blue
blue = AveragePooling2D(pool_size=(4,4),name='blue_pool')(base)
blue = Convolution2D(filters=64,kernel_size=(1,1),name='blue_1_by_1')(blue)
blue = UpSampling2D(size=4,interpolation='bilinear',name='blue_upsampling')(blue)
# green
green = AveragePooling2D(pool_size=(8,8),name='green_pool')(base)
green = Convolution2D(filters=64,kernel_size=(1,1),name='green_1_by_1')(green)
green = UpSampling2D(size=8,interpolation='bilinear',name='green_upsampling')(green)
# base + red + yellow + blue + green
return tf.keras.layers.concatenate([base,red,yellow,blue,green])

```

```

def last_conv_module(input_layer):
    X = pyramid_feature_maps(input_layer)
    X = Convolution2D(filters=1,kernel_size=1,padding='same',name='last_conv_3_by_3')(X)
    #X = BatchNormalization(name='last_conv_3_by_3_batch_norm')(X)
    X = Activation('sigmoid',name='last_conv_sigmoid')(X)
    #X = tf.keras.layers.Flatten(name='last_conv_flatten')(X)
    return X

```

last_conv_3_by_3 (Conv2D)	(None, 256, 256, 1)	513	concatenate_2[0][0]
last_conv_sigmoid (Activation)	(None, 256, 256, 1)	0	last_conv_3_by_3[0][0]
<hr/>			
Total params: 700,513			
Trainable params: 697,825			
Non-trainable params: 2,688			

4.6.2.1 L'entraînement du modèle

```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-5),
              loss=dice_loss,
              metrics=[dice_coef, iou, Recall(), Precision()])
```

```
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint

checkpointer = ModelCheckpoint(filepath="PSP_Model-weights-6.hdf5",
                               verbose=1,
                               save_best_only=True,
                               save_weights_only=True
                               )
reduce_lr = ReduceLROnPlateau(monitor='val_loss',
                             mode='min',
                             verbose=1,
                             patience=5,
                             min_delta=0.0001,
                             factor=0.2
                             )
```

Pour entraîner le modèle PSPNet, nous avons configuré le processus en utilisant l'optimiseur Adam avec un taux d'apprentissage de $1e-5$ et une fonction de perte basée sur le coefficient Dice. Pour surveiller et ajuster dynamiquement l'apprentissage, nous avons intégré des rappels essentiels. Le rappel ModelCheckpoint a été mis en place pour sauvegarder les poids du modèle à chaque itération, ne conservant que la meilleure version basée sur les performances sur les données de validation. Parallèlement, le rappel ReduceLROnPlateau a été configuré pour surveiller la perte sur les données de validation, diminuant le taux d'apprentissage de 20% si aucune amélioration n'est observée pendant cinq itérations consécutives. Ces mécanismes de contrôle dynamique garantissent une convergence efficace du modèle, optimisant ainsi ses performances et sa capacité à saisir des informations contextuelles à différentes échelles spatiales lors de la segmentation d'images complexes, telles que les images médicales et les scènes extérieures. En utilisant ces stratégies de rappel en tandem avec les paramètres d'entraînement spécifiés, le PSPNet est formé pour offrir une segmentation précise et détaillée tout en s'adaptant à la diversité des données.

4.6.2.2 Les résultats de l'entraînement

```

Epoch 1/100
263/263 [=====] - 156s 593ms/step - loss: 0.7510 - dice_coef: 0.2490 - iou: 0.1595 - r
ecall_1: 0.7748 - precision_1: 0.1289 - val_loss: 0.4182 - val_dice_coef: 0.5673 - val_iou: 0.4123 - val_recall
_1: 0.6560 - val_precision_1: 0.6380

Epoch 00001: val_loss improved from inf to 0.41823, saving model to PSP_Model-weights-6.hdf5
Epoch 2/100
263/263 [=====] - 155s 591ms/step - loss: 0.3935 - dice_coef: 0.6065 - iou: 0.4566 - r
ecall_1: 0.5983 - precision_1: 0.7224 - val_loss: 0.3713 - val_dice_coef: 0.6130 - val_iou: 0.4604 - val_recall
_1: 0.6509 - val_precision_1: 0.6746

Epoch 00002: val_loss improved from 0.41823 to 0.37132, saving model to PSP_Model-weights-6.hdf5

Epoch 99/100
263/263 [=====] - 155s 590ms/step - loss: 0.2118 - dice_coef: 0.7882 - iou: 0.6596 - r
ecall_1: 0.7736 - precision_1: 0.8508 - val_loss: 0.2284 - val_dice_coef: 0.7523 - val_iou: 0.6265 - val_recall
_1: 0.8013 - val_precision_1: 0.8172

Epoch 00099: val_loss did not improve from 0.22796
Epoch 100/100
263/263 [=====] - 155s 591ms/step - loss: 0.2085 - dice_coef: 0.7915 - iou: 0.6657 - r
ecall_1: 0.7732 - precision_1: 0.8657 - val_loss: 0.2289 - val_dice_coef: 0.7519 - val_iou: 0.6259 - val_recall
_1: 0.8001 - val_precision_1: 0.8179

Epoch 00100: val_loss did not improve from 0.22796

```

Au cours de l'entraînement du modèle PSPNet sur 100 epochs, des améliorations significatives ont été observées dans ses performances. Initialement, la perte moyenne sur les données d'entraînement était élevée, atteignant 0.7510, avec un coefficient Dice de 0.2490 et un IoU de 0.1595. Cependant, au fil de l'entraînement, une nette amélioration a été constatée, avec une réduction notable de la perte moyenne à 0.2085 à la 100e epoch. Cette amélioration s'accompagne d'une nette augmentation du coefficient Dice à 0.7915 et de l'IoU à 0.6657. Les résultats sur les données de validation ont également témoigné de cette progression positive, passant d'une perte moyenne de 0.41823 à 0.2289, avec une amélioration conséquente du coefficient Dice de 0.5673 à 0.7519 et de l'IoU de 0.4123 à 0.6259. Ces résultats démontrent la capacité du modèle à apprendre des structures spatiales complexes tout en améliorant significativement sa précision de segmentation au fil de l'entraînement, offrant ainsi des perspectives encourageantes pour une segmentation plus précise des images médicales.

Voici ce graphique illustrant ces métriques de performance ainsi que la fonction de coût (Dice Loss) tout au long de 100 époques :

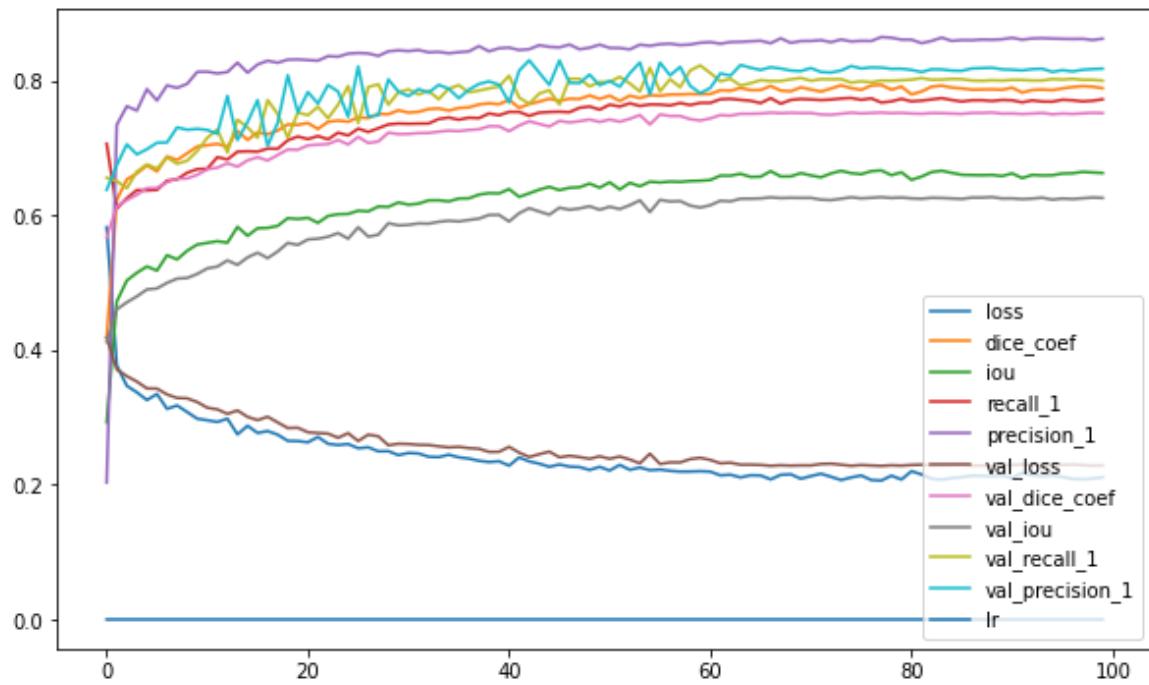


FIGURE 4.7: Synthèse de la Performance du Modèle PSPNet : Données d'Entraînement et de Validation

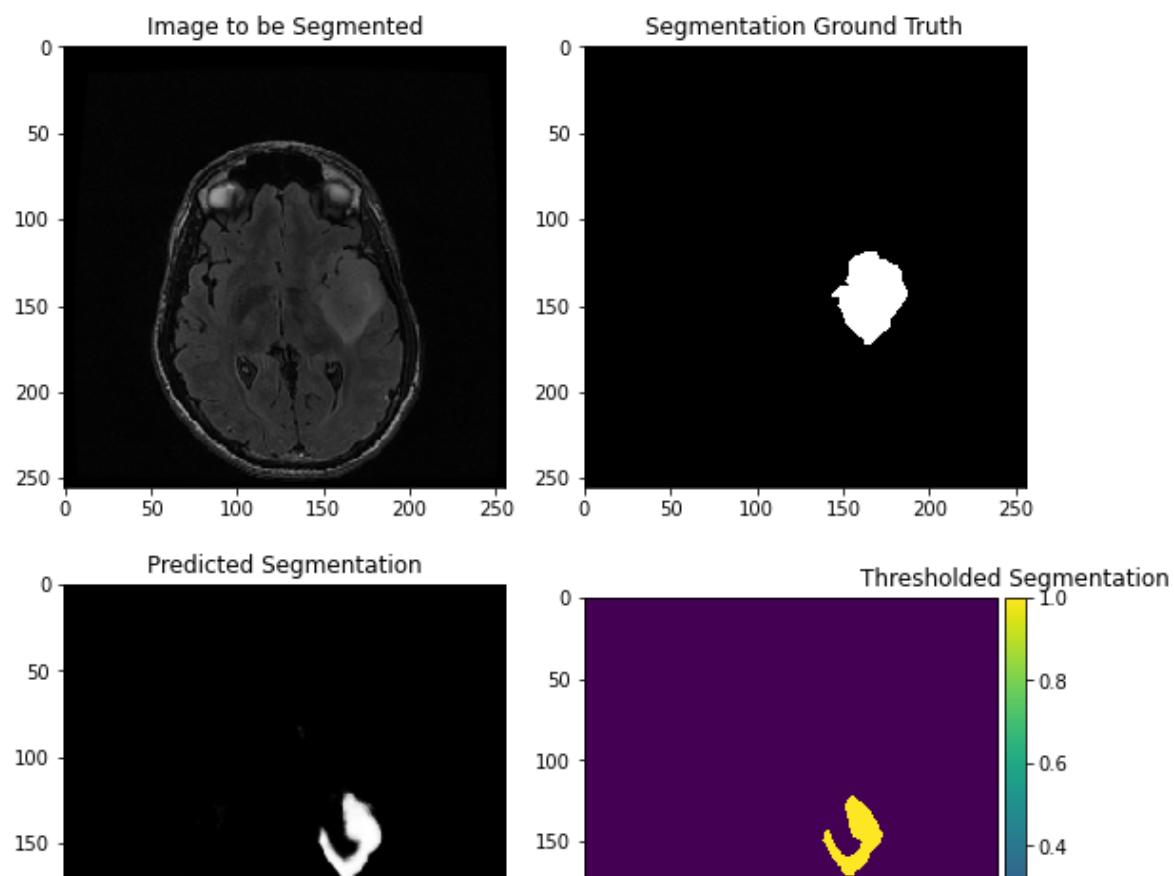
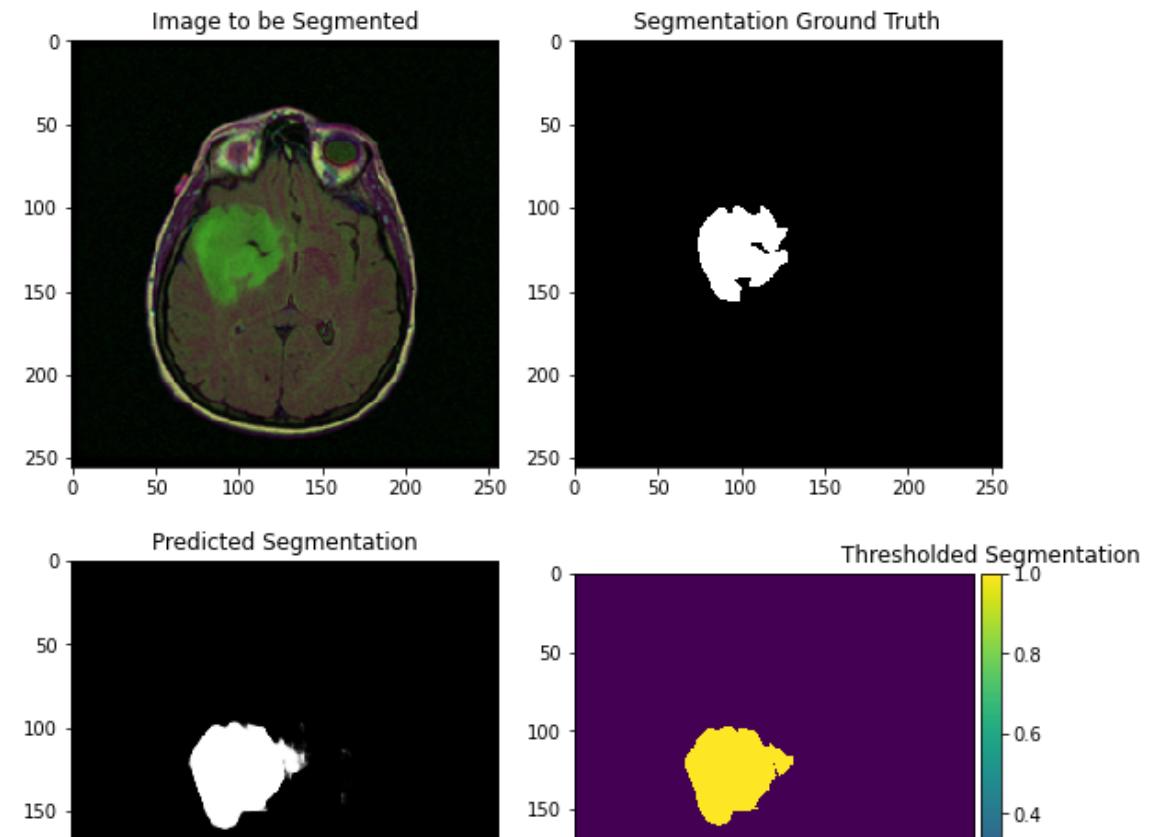
4.6.2.3 L'évaluation du modèle

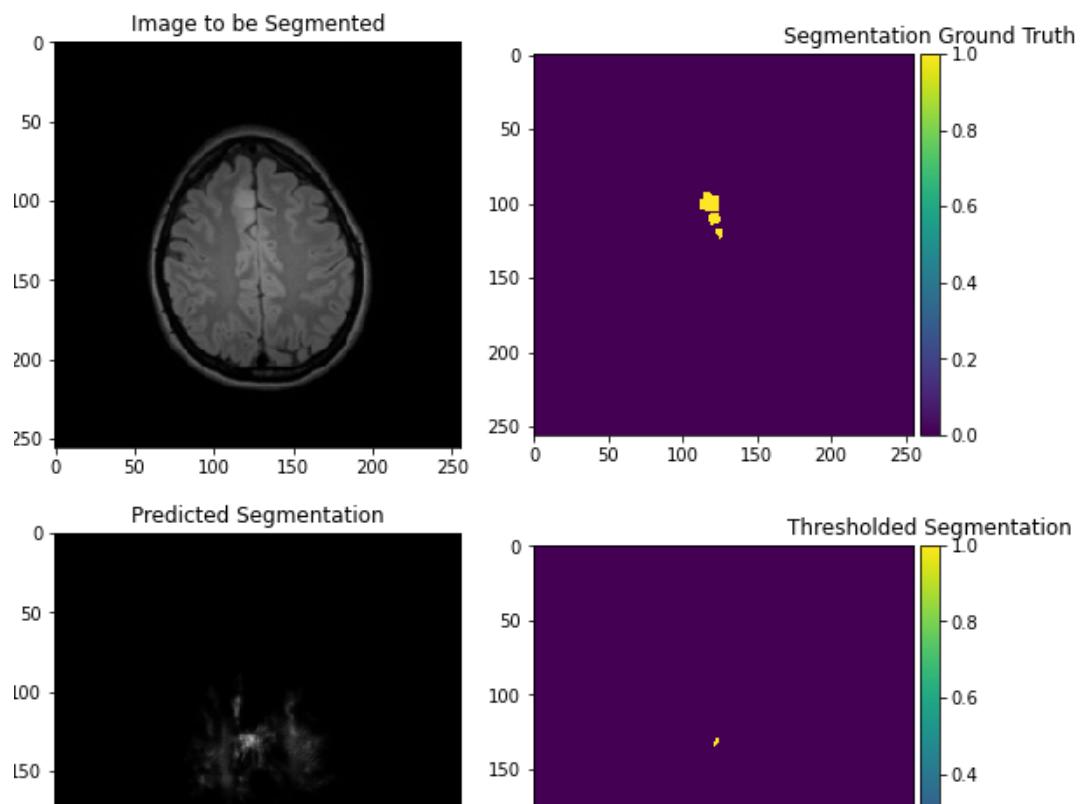
Ces résultats d'évaluation sur les données de test suggèrent que le modèle PSPNet performe plutôt bien dans la segmentation des tumeurs cérébrales. Avec un coefficient Dice de 0.7732 et un IoU de 0.6393, le modèle semble capturer les détails essentiels dans les images. De plus, un rappel de 0.7954 et une précision de 0.8086 indiquent une capacité prometteuse à identifier les régions d'intérêt avec une précision satisfaisante. Ces performances solides peuvent contribuer à une analyse précise des données d'imagerie médicale, soulignant ainsi la fiabilité du modèle pour cette tâche spécifique.

```
model.load_weights("/kaggle/input/pspnet/PSP_Model-weights-6.hdf5")
print("Evaluate on test data")
results = model.evaluate(X_test, y_test.astype(np.float32), batch_size=4)

52/52 [=====] - 10s 181ms/step - loss: 0.2268 - dice_coef: 0.7732 - iou: 0.6393 - recall_1: 0.7954 - precision_1: 0.8086
```

4.6.2.4 Tester le modèle sur les images du test





4.6.3 SegNet

L'implémentation de SegNet représente un jalon essentiel dans notre projet axé sur la segmentation sémantique. SegNet se distingue par son architecture spécialisée encodeur-décodeur. Cette approche vise à capturer des informations sémantiques tout en préservant les détails spatiaux, caractéristiques cruciales pour une segmentation précise. L'utilisation du pooling non paramétrique et d'autres techniques innovantes confère à SegNet la capacité de traiter efficacement des images complexes, en attribuant des étiquettes sémantiques à chaque pixel.

Voici le modèle implémenté :

```

def segnet(epochs_num, savename):

    # Encoding layer
    img_input = Input(shape= (256, 256, 3))
    x = Conv2D(64, (3, 3), padding='same', name='conv1', strides= (1,1))(img_input)
    x = BatchNormalization(name='bn1')(x)
    x = Activation('relu')(x)
    x = Conv2D(64, (3, 3), padding='same', name='conv2')(x)
    x = BatchNormalization(name='bn2')(x)
    x = Activation('relu')(x)
    x = MaxPooling2D()(x)

    x = Conv2D(128, (3, 3), padding='same', name='conv3')(x)
    x = BatchNormalization(name='bn3')(x)
    x = Activation('relu')(x)
    x = Conv2D(128, (3, 3), padding='same', name='conv4')(x)
    x = BatchNormalization(name='bn4')(x)

```

```

    x = Activation('relu')(x)
    x = MaxPooling2D()(x)

    x = Conv2D(256, (3, 3), padding='same', name='conv5')(x)
    x = BatchNormalization(name='bn5')(x)
    x = Activation('relu')(x)
    x = Conv2D(256, (3, 3), padding='same', name='conv6')(x)
    x = BatchNormalization(name='bn6')(x)
    x = Activation('relu')(x)
    x = Conv2D(256, (3, 3), padding='same', name='conv7')(x)
    x = BatchNormalization(name='bn7')(x)
    x = Activation('relu')(x)
    x = MaxPooling2D()(x)

    x = Conv2D(512, (3, 3), padding='same', name='conv8')(x)
    x = BatchNormalization(name='bn8')(x)
    x = Activation('relu')(x)
    x = Conv2D(512, (3, 3), padding='same', name='conv9')(x)

```

```

x = BatchNormalization(name='bn9')(x)
x = Activation('relu')(x)
x = Conv2D(512, (3, 3), padding='same', name='conv10')(x)
x = BatchNormalization(name='bn10')(x)
x = Activation('relu')(x)
x = MaxPooling2D()(x)

x = Conv2D(512, (3, 3), padding='same', name='conv11')(x)
x = BatchNormalization(name='bn11')(x)
x = Activation('relu')(x)
x = Conv2D(512, (3, 3), padding='same', name='conv12')(x)
x = BatchNormalization(name='bn12')(x)
x = Activation('relu')(x)
x = Conv2D(512, (3, 3), padding='same', name='conv13')(x)
x = BatchNormalization(name='bn13')(x)
x = Activation('relu')(x)
x = MaxPooling2D()(x)

```

```

x = Dense(1024, activation = 'relu', name='fc1')(x)
x = Dense(1024, activation = 'relu', name='fc2')(x)
# Decoding Layer
x = UpSampling2D()(x)
x = Conv2DTranspose(512, (3, 3), padding='same', name='deconv1')(x)
x = BatchNormalization(name='bn14')(x)
x = Activation('relu')(x)
x = Conv2DTranspose(512, (3, 3), padding='same', name='deconv2')(x)
x = BatchNormalization(name='bn15')(x)
x = Activation('relu')(x)
x = Conv2DTranspose(512, (3, 3), padding='same', name='deconv3')(x)
x = BatchNormalization(name='bn16')(x)
x = Activation('relu')(x)

x = UpSampling2D()(x)
x = Conv2DTranspose(512, (3, 3), padding='same', name='deconv4')(x)
x = BatchNormalization(name='bn17')(x)
x = Activation('relu')(x)

```

```

x = Conv2DTranspose(512, (3, 3), padding='same', name='deconv5')(x)
x = BatchNormalization(name='bn18')(x)
x = Activation('relu')(x)
x = Conv2DTranspose(256, (3, 3), padding='same', name='deconv6')(x)
x = BatchNormalization(name='bn19')(x)
x = Activation('relu')(x)

x = UpSampling2D()(x)
x = Conv2DTranspose(256, (3, 3), padding='same', name='deconv7')(x)
x = BatchNormalization(name='bn20')(x)
x = Activation('relu')(x)
x = Conv2DTranspose(256, (3, 3), padding='same', name='deconv8')(x)
x = BatchNormalization(name='bn21')(x)
x = Activation('relu')(x)
x = Conv2DTranspose(128, (3, 3), padding='same', name='deconv9')(x)
x = BatchNormalization(name='bn22')(x)
x = Activation('relu')(x)

```

```

x = UpSampling2D()(x)
x = Conv2DTranspose(128, (3, 3), padding='same', name='deconv10')(x)
x = BatchNormalization(name='bn23')(x)
x = Activation('relu')(x)
x = Conv2DTranspose(64, (3, 3), padding='same', name='deconv11')(x)
x = BatchNormalization(name='bn24')(x)
x = Activation('relu')(x)

x = UpSampling2D()(x)
x = Conv2DTranspose(64, (3, 3), padding='same', name='deconv12')(x)
x = BatchNormalization(name='bn25')(x)
x = Activation('relu')(x)
x = Conv2DTranspose(1, (3, 3), padding='same', name='deconv13')(x)
x = BatchNormalization(name='bn26')(x)
x = Activation('sigmoid')(x)
pred = Reshape((256,256))(x)

model = Model(inputs=img_input, outputs=pred)

```

4.6.3.1 L'entraînement du modèle

```

    model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.001, mo
mentum=0.9, nesterov=False),
                  loss=dice_loss, metrics=[iou, dice_coef, Precision(), Recall
()])
    model.summary()
    hist = model.fit(train_data, epochs= epochs_num, batch_size= 20, valida
tion_data= val_data, verbose=1)

    model.save(savename)
    return model,hist

```

Pendant la phase d'entraînement du modèle SegNet, l'architecture du réseau est entraînée sur l'ensemble de données en utilisant l'algorithme de descente de gradient stochastique (SGD) avec un taux d'apprentissage de 0.001 et un momentum de 0.9. La fonction de perte binaire croisée est employée pour quantifier l'écart entre les prédictions du modèle et les annotations réelles. Le processus d'entraînement se déroule sur plusieurs époques, chaque époque représentant une itération complète à travers l'ensemble de données. Des métriques de performance telles que l'indice Jaccard, le coefficient de Dice, la précision, le rappel et l'exactitude sont calculées à chaque itération pour évaluer la qualité de la segmentation obtenue. La progression de l'entraînement est surveillée à l'aide de données de validation. Une fois l'entraînement terminé, le modèle résultant est sauvegardé sous un nom spécifié, prêt à être utilisé pour des prédictions sur de nouvelles données.

4.6.3.2 Les résultats de l'entraînement

```

Epoch 1/100
208/208 [=====] - 178s 709ms/step - loss: 0.972
0 - iou: 0.0143 - dice_coef: 0.0280 - precision: 0.0183 - recall: 0.7384
- val_loss: 0.9817 - val_iou: 0.0092 - val_dice_coef: 0.0183 - val_preci
sion: 0.0000e+00 - val_recall: 0.0000e+00
Epoch 2/100
208/208 [=====] - 138s 663ms/step - loss: 0.961
1 - iou: 0.0199 - dice_coef: 0.0389 - precision: 0.0321 - recall: 0.9838
- val_loss: 0.9624 - val_iou: 0.0193 - val_dice_coef: 0.0376 - val_preci
sion: 0.0476 - val_recall: 0.9435

```

```
Epoch 99/100
208/208 [=====] - 139s 668ms/step - loss: 0.100
9 - iou: 0.8222 - dice_coef: 0.8991 - precision: 0.9206 - recall: 0.9237
- val_loss: 0.2175 - val_iou: 0.6498 - val_dice_coef: 0.7825 - val_precision: 0.8916 - val_recall: 0.7543
Epoch 100/100
208/208 [=====] - 139s 668ms/step - loss: 0.089
4 - iou: 0.8410 - dice_coef: 0.9106 - precision: 0.9257 - recall: 0.9321
- val_loss: 0.1826 - val_iou: 0.7009 - val_dice_coef: 0.8174 - val_precision: 0.8580 - val_recall: 0.8423
```

Le processus d'entraînement du modèle de segmentation a été effectué sur 100 epochs, avec une observation des métriques clés tout au long du processus. Au début de l'entraînement (Epoch 1), le modèle présentait une perte (loss) de 0.9720 avec un faible score d'intersection over union (IOU) de 0.0143 et un coefficient Dice de 0.0280. Les mesures de précision (precision) et de rappel (recall) étaient également relativement faibles à 0.0183 et 0.7384 respectivement pour cette epoch.

Au fur et à mesure de l'entraînement, on observe une amélioration significative dans les performances du modèle. À l'Epoch 100, la perte a été réduite à 0.0894, indiquant une meilleure convergence du modèle. Les métriques d'IOU et de coefficient Dice se sont considérablement améliorées, atteignant respectivement 0.8410 et 0.9106, témoignant ainsi d'une meilleure précision dans la segmentation des tumeurs cérébrales. De plus, la précision et le rappel ont significativement augmenté, atteignant des scores de 0.9257 et 0.9321 respectivement à la fin de l'entraînement.

Les valeurs de validation (val loss, val iou, val dice coef, val precision, val recall) pour l'ensemble de validation suivent une tendance similaire, confirmant la capacité croissante du modèle à généraliser et à segmenter efficacement les régions d'intérêt sur de nouvelles données. Ces améliorations successives soulignent l'efficacité de l'entraînement du modèle dans la segmentation précise des tumeurs cérébrales, démontrant ainsi sa capacité à améliorer les diagnostics cliniques dans le domaine de l'imagerie médicale surtout la segmentations des tumeurs cérébrales.

Le graphique ci-dessous présente l'évolution des métriques de performance sur les 100 époques d'entraînement du modèle SegNet. Cette visualisation offre un aperçu de la progression de la performance du modèle au fil du temps, mettant en évidence les tendances d'amélioration et de convergence des métriques clés au cours du processus d'optimisation.

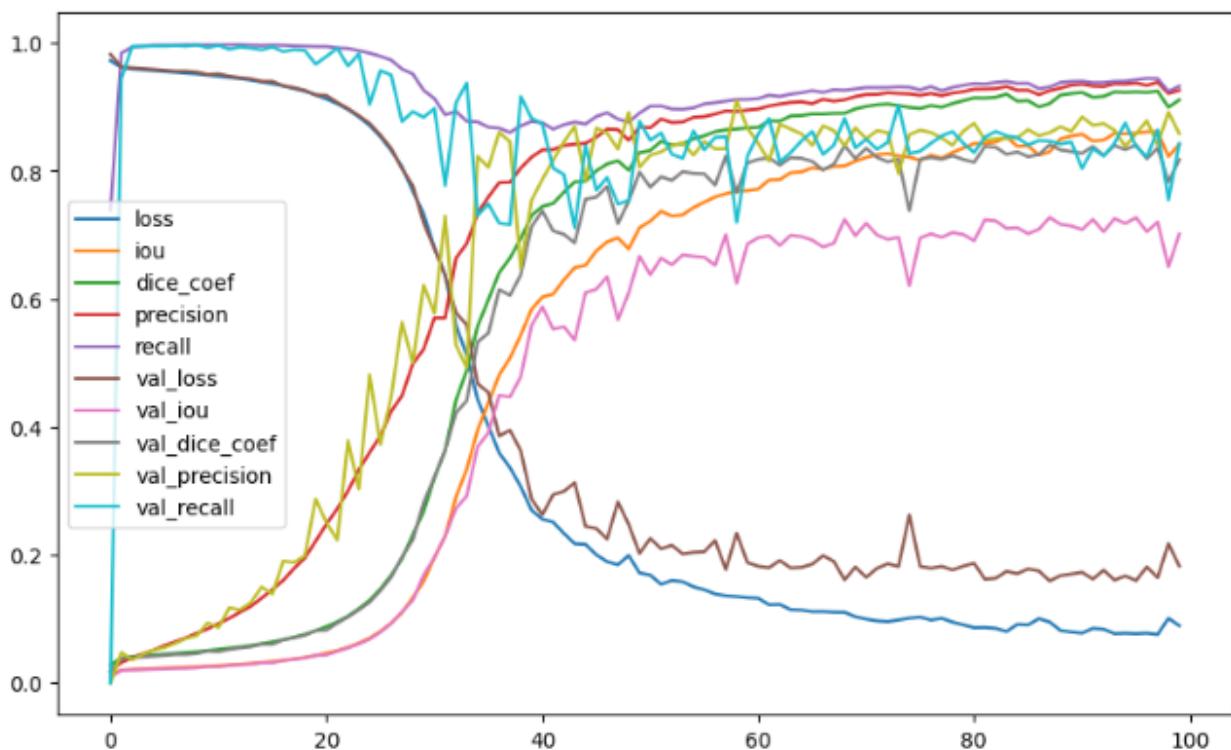


FIGURE 4.8: Synthèse de la Performance du Modèle SegNet : Données d’Entraînement et de Validation

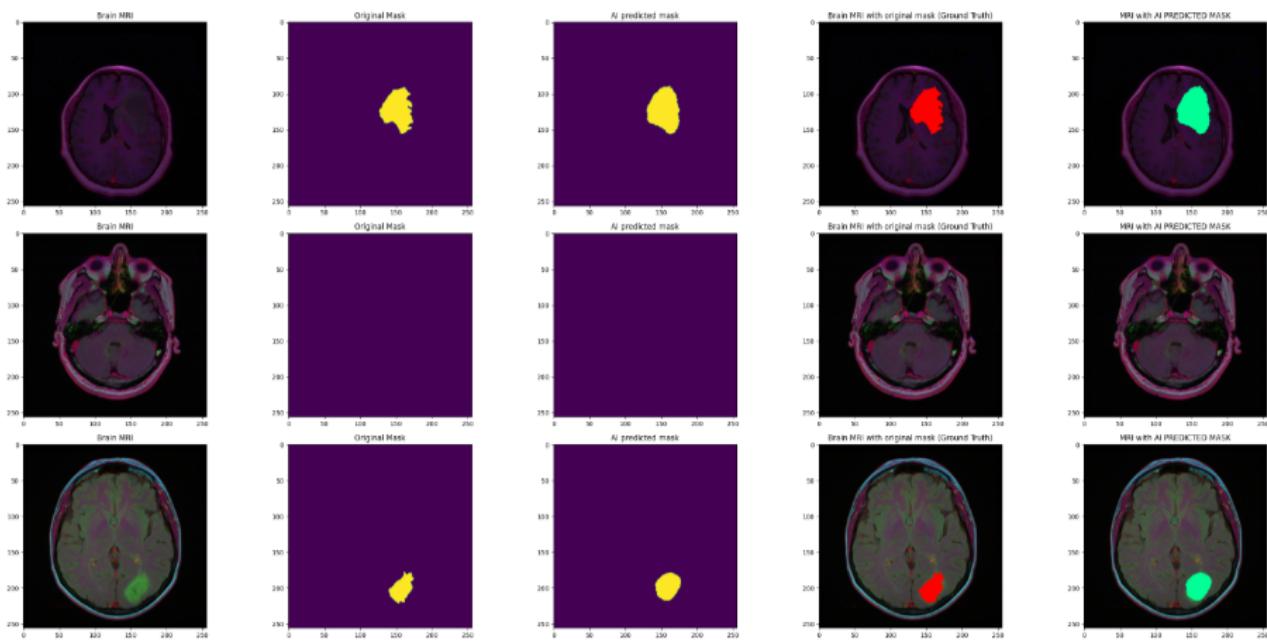
4.6.3.3 L'évaluation du modèle

Cette évaluation post-entraînement du modèle de segmentation montre des performances prometteuses. Les métriques d'intersection over union (IOU) et de coefficient Dice dépassent les 70%, indiquant une bonne précision dans la segmentation des tumeurs cérébrales sur de nouvelles données. De plus, les scores de précision et de rappel approchent respectivement les 80% et les 90%, soulignant la capacité du modèle à identifier précisément les zones d'intérêt et à rappeler efficacement les régions tumorales dans les images IRM. Ces résultats sont encourageants quant à l'efficacité du modèle pour la segmentation précise des tumeurs cérébrales.

```
test_ids = list(test.image_path)
test_mask= list(test.mask_path)
test_data = DataGenerator(test_ids, test_mask)
tv = model.evaluate(test_data)
print("Segmentation iou is {:.2f}%".format(tv[1]*100))
print("Segmentation dice_coef is {:.2f}%".format(tv[2]*100))
print("Segmentation precision is {:.2f}%".format(tv[3]*100))
print("Segmentation recall is {:.2f}%".format(tv[4]*100))
```

```
18/18 [=====] - 4s 221ms/step - loss: 0.1565 - iou: 0.7323 - dice_coef: 0.8435 - precision: 0.8269 - recall: 0.8902
Segmentation iou is 73.23%
Segmentation dice_coef is 84.35%
Segmentation precision is 82.69%
Segmentation recall is 89.02%
```

4.6.3.4 Tester le modèle sur les images du test



4.6.4 UNet

Après avoir décrit l'architecture de UNet, nous passons maintenant à son implémentation pratique. Le code suivant illustre la mise en œuvre de l'architecture UNet en utilisant la bibliothèque Keras, un choix populaire pour les applications de deep learning en raison de sa flexibilité et de son efficacité. Voici le modèle implémenté :

```
def unet(pretrained_weights = None, input_size = (256,256,3)):
    inputs = Input(input_size)
    conv1 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(inputs)
    conv1 = Conv2D(64, 3, activation = None, padding = 'same', kernel_initializer = 'he_normal')(conv1)
    bn_1 = BatchNormalization(axis=3)(conv1)
    act_1 = Activation('relu')(bn_1)
    pool1 = MaxPooling2D(pool_size=(2, 2))(act_1)
    conv2 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(pool1)
    conv2 = Conv2D(128, 3, activation = None, padding = 'same', kernel_initializer = 'he_normal')(conv2)
    bn_2 = BatchNormalization(axis=3)(conv2)
    act_2 = Activation('relu')(bn_2)
    pool2 = MaxPooling2D(pool_size=(2, 2))(act_2)
    conv3 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(pool2)
    conv3 = Conv2D(256, 3, activation = None, padding = 'same', kernel_initializer = 'he_normal')(conv3)
    bn_3 = BatchNormalization(axis=3)(conv3)
    act_3 = Activation('relu')(bn_3)
    pool3 = MaxPooling2D(pool_size=(2, 2))(act_3)
    conv4 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(pool3)
    conv4 = Conv2D(512, 3, activation = None, padding = 'same', kernel_initializer = 'he_normal')(conv4)
    bn_4 = BatchNormalization(axis=3)(conv4)
    act_4 = Activation('relu')(bn_4)
    pool4 = MaxPooling2D(pool_size=(2, 2))(act_4)
```

```

conv5 = Conv2D(1024, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(pool4)
conv5 = Conv2D(1024, 3, activation = None, padding = 'same', kernel_initializer = 'he_normal')(conv5)
bn_5 = BatchNormalization(axis=3)(conv5)
act_5 = Activation('relu')(bn_5)
up6 = Conv2D(512, 2, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(UpSampling2D(size = (2,2))(act_5))
merge6 = concatenate([conv4,up6], axis = 3)
conv6 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(merge6)
conv6 = Conv2D(512, 3, activation = None, padding = 'same', kernel_initializer = 'he_normal')(conv6)
bn_6 = BatchNormalization(axis=3)(conv6)
act_6 = Activation('relu')(bn_6)

up7 = Conv2D(256, 2, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(UpSampling2D(size = (2,2))(act_6))
merge7 = concatenate([conv3,up7], axis = 3)
conv7 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(merge7)
conv7 = Conv2D(256, 3, activation = None, padding = 'same', kernel_initializer = 'he_normal')(conv7)
bn_7 = BatchNormalization(axis=3)(conv7)
act_7 = Activation('relu')(bn_7)
up8 = Conv2D(128, 2, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(UpSampling2D(size = (2,2))(act_7))
merge8 = concatenate([conv2,up8], axis = 3)
conv8 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(merge8)
conv8 = Conv2D(128, 3, activation = None, padding = 'same', kernel_initializer = 'he_normal')(conv8)
bn_8 = BatchNormalization(axis=3)(conv8)
act_8 = Activation('relu')(bn_8)

up9 = Conv2D(64, 2, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(UpSampling2D(size = (2,2))(act_8))
merge9 = concatenate([conv1,up9], axis = 3)
conv9 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(merge9)
conv9 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv9)
conv9 = Conv2D(2, 3, activation = None, padding = 'same', kernel_initializer = 'he_normal')(conv9)
bn_9 = BatchNormalization(axis=3)(conv9)
act_9 = Activation('relu')(bn_9)

conv10 = Conv2D(1, 1, activation = 'sigmoid')(act_9)
model = model = Model(inputs=[inputs] , outputs = [conv10])
    
```

4.6.4.1 L'entraînement du modèle

Pendant la phase d'entraînement du modèle U-Net, l'architecture du réseau est entraînée sur l'ensemble de données en utilisant optimiseur Adam avec un taux d'apprentissage de 0.001. La fonction de perte dice est employée pour quantifier l'écart entre les prédictions du modèle et les annotations réelles. Le processus d'entraînement se déroule sur plusieurs époques, chaque époque représentant une itération complète à travers l'ensemble de données. Des métriques de performance telles que l'indice Jaccard, le coefficient de Dice, la précision, le rappel et l'exactitude sont calculées à chaque itération pour évaluer la qualité de la segmentation obtenue. La progression de l'entraînement est surveillée à l'aide de données de validation. Une fois l'entraînement terminé, le modèle résultant est sauvegardé sous un nom spécifié, prêt à être utilisé pour des prédictions sur de nouvelles données.

```
from tensorflow.keras.metrics import Recall, Precision

EPOCH = 100
learning_rate = 1e-3

model.compile(optimizer=Adam(learning_rate=1e-3, beta_1=0.9, beta_2=0.999, epsilon=None,
                            decay=1e-3/32, amsgrad=False),
              loss=dice_coef_loss, metrics=[dice_coef, iou, Recall(), Precision()])
```

X Hide code

```
history = model.fit(train_gen, steps_per_epoch=len(mri_train)/32,
                     epochs=EPOCH, validation_data=val_gen, validation_steps=len(mri_val) / 32,
                     callbacks=[model_checkpoint])
```

4.6.4.2 Les résultats de l'entraînement

Au cours de l'entraînement du modèle U-Net sur 100 epochs, des améliorations significatives ont été observées dans ses performances. On observe une amélioration constante des valeurs de précision, de rappel (recall), et du coefficient de Dice (dice coef), ainsi que la réduction de la perte de validation (val loss) au fur et à mesure que l'entraînement progresse. Notamment, l'epoch 98 montre une précision de 89.70 et un rappel de 88.57, signifiant une forte pertinence dans les prédictions du modèle avec une capacité notable à retrouver toutes les instances pertinentes. Cependant, il est à noter que l'augmentation de certaines métriques telles que la précision n'entraîne pas toujours des améliorations dans d'autres domaines, comme en témoigne l'epoch 99 où la perte de validation n'a pas amélioré le précédent meilleur score. Au commencement de l'entraînement, l'epoch 1 révèle un coefficient de Dice de 0.8361, qui s'est progressivement amélioré, soulignant l'efficacité de l'apprentissage du modèle. L'amélioration continue au cours des epochs subséquentes est évidente, avec l'epoch 2 enregistrant déjà un coefficient de Dice de 0.8389.

Ces tendances positives démontrent la capacité du modèle à généraliser à partir des données d'entraînement tout en minimisant le surajustement, une réalisation essentielle pour l'applicabilité pratique du modèle U-Net dans des situations réelles.

```

Found 2828 validated image filenames.
Found 2828 validated image filenames.
Epoch 1/100
89/88 [=====] - ETA: 0s - loss: 0.9639 - dice_coef: 0.0361 - iou: 0.018
4 - recall: 0.9213 - precision: 0.0759
Found 708 validated image filenames.
Found 708 validated image filenames.
88/88 [=====] - 128s 1s/step - loss: 0.9639 - dice_coef: 0.0361 - iou:
0.0184 - recall: 0.9213 - precision: 0.0759 - val_loss: 0.9710 - val_dice_coef: 0.0291 - val_i
o: 0.0148 - val_recall: 0.5676 - val_precision: 0.2236

Epoch 00001: val_iou improved from inf to 0.01484, saving model to unet_membrane.hdf5
Epoch 2/100
88/88 [=====] - 97s 1s/step - loss: 0.9612 - dice_coef: 0.0389 - iou:
0.0199 - recall: 0.9235 - precision: 0.1241 - val_loss: 0.9663 - val_dice_coef: 0.0325 - val_i
o: 0.0166 - val_recall: 0.7186 - val_precision: 0.0612

Epoch 98/100
88/88 [=====] - 97s 1s/step - loss: 0.1433 - dice_coef: 0.8572 - iou:
0.7565 - recall: 0.8857 - precision: 0.8970 - val_loss: 0.1421 - val_dice_coef: 0.8601 - val_i
o: 0.7574 - val_recall: 0.9089 - val_precision: 0.8757

Epoch 00098: val_iou did not improve from 0.01409
Epoch 99/100
88/88 [=====] - 97s 1s/step - loss: 0.1317 - dice_coef: 0.8671 - iou:
0.7720 - recall: 0.9047 - precision: 0.9053 - val_loss: 0.1539 - val_dice_coef: 0.8482 - val_i
o: 0.7394 - val_recall: 0.9093 - val_precision: 0.8614

Epoch 00099: val_iou did not improve from 0.01409
Epoch 100/100
88/88 [=====] - 97s 1s/step - loss: 0.1250 - dice_coef: 0.8748 - iou:
0.7818 - recall: 0.9092 - precision: 0.9030 - val_loss: 0.1334 - val_dice_coef: 0.8696 - val_i
o: 0.7719 - val_recall: 0.8834 - val_precision: 0.9105

```



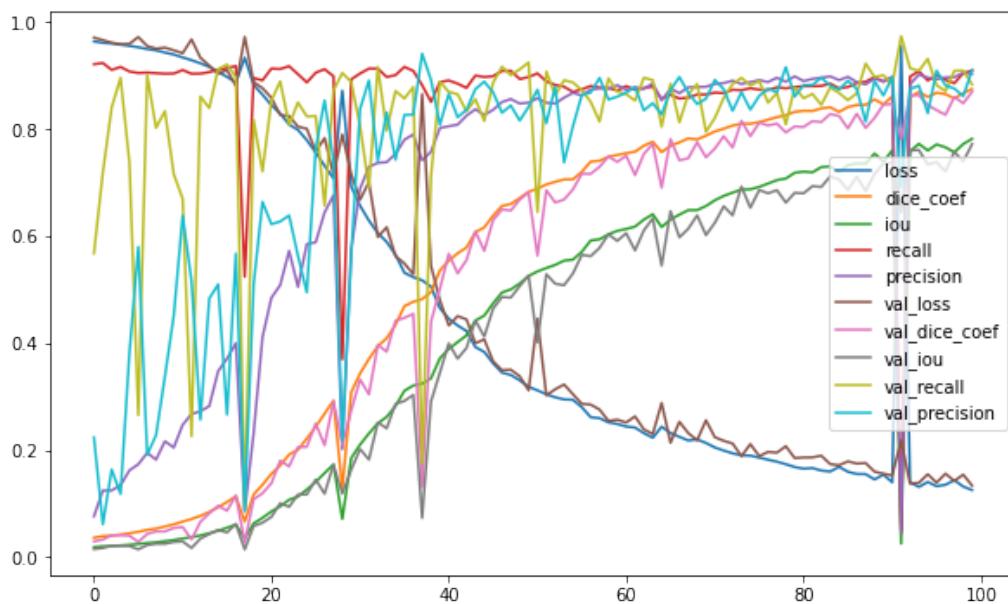
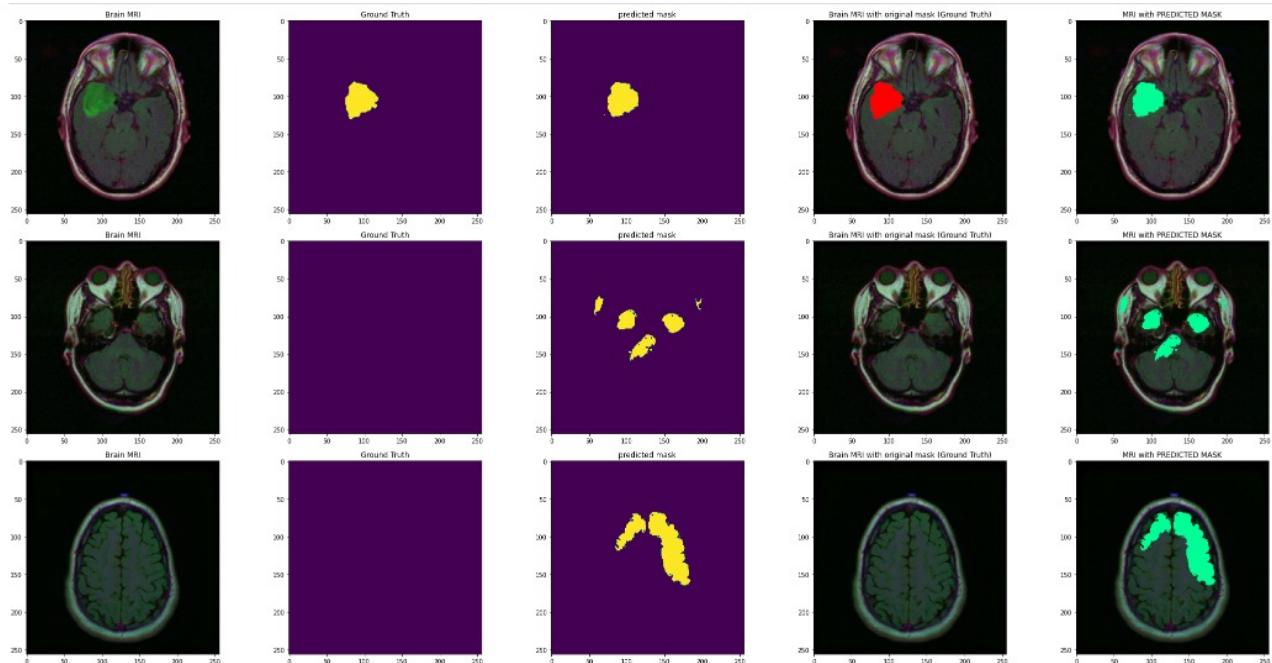


FIGURE 4.9: Synthèse de la Performance du Modèle Unet : Données d’Entraînement et de Validation

4.6.4.3 L'évaluation du modèle



Les résultats d'évaluation du modèle UNet sur la partie du test pour segmentation des tumeurs cérébrales indiquent une performance globalement positive, bien que certains aspects nécessitent une analyse plus approfondie pour mieux comprendre les capacités et les limites du modèle. Voici une synthèse des principales observations basées sur les métriques fournies :

1. **Perte :** La perte globale sur l'ensemble de test est relativement basse (0.1337), suggérant une minimisation efficace de l'erreur pendant la phase d'apprentissage.

2. **Coefficient de Dice :** Le coefficient de Dice sur l'ensemble de test est notable (0.8396), indiquant une bonne concordance entre les prédictions du modèle et la vérité terrain. Cela témoigne d'une capacité précise du modèle UNet à délimiter les contours des tumeurs cérébrales.
3. **Intersection sur l'Union (IoU) :** L'indice Jaccard (IoU) sur l'ensemble de test (0.7217) suggère une bonne superposition entre les prédictions du modèle et la vérité terrain. Une analyse plus approfondie pourrait être nécessaire pour évaluer la manière dont le modèle traite spécifiquement les zones de chevauchement.
4. **Rappel :** Le rappel sur l'ensemble de test est élevé (0.8234), indiquant que le modèle UNet détecte de manière sensible les tumeurs cérébrales présentes dans les images.
5. **Précision :** La précision sur l'ensemble de test (0.910) est élevée, suggérant que le modèle minimise efficacement les fausses prédictions positives. Cela indique que les régions prédites comme tumeurs par le modèle sont souvent correctes.

En résumé, le modèle UNet semble bien réussir dans la tâche de segmentation des tumeurs cérébrales, avec une précision, un rappel élevés, et une bonne concordance globale avec la vérité terrain.

4.6.5 Attention-UNet

Les caractéristiques distinctives du modèle Attention UNet se manifestent par sa remarquable capacité à capturer des détails minutieux tout en préservant une précision de segmentation exceptionnelle, même pour des objets de petite taille au sein des images. Son architecture repose sur une fusion innovante de plusieurs éléments clés, mettant en avant l'intégration astucieuse d'un mécanisme d'attention. Ce dernier permet au modèle de se concentrer de manière sélective sur des zones spécifiques de l'image, améliorant significativement sa capacité à saisir des informations sémantiques tout en conservant les détails spatiaux essentiels. L'utilisation de l'attention dans le modèle Attention UNet représente une stratégie raffinée pour la segmentation sémantique, permettant une analyse détaillée à différentes échelles au sein de l'image.

Voici le modèle implémenté :

```
from tensorflow.keras.layers import Conv2D, BatchNormalization, Activation,
MaxPool2D, Conv2DTranspose, Concatenate, Input
from tensorflow.keras.models import Model
from tensorflow.keras.applications import ResNet50
# #lambda function for repeating the result from AG
def repeat_elem(tensor, rep):
    return tf.keras.layers.Lambda(lambda x, repnum: K.repeat_elements(x, r
epnum, axis=3), arguments={'repnum': rep})(tensor)
# Attention Gate
def attention_gate(g, s, num_filters):
    Wg=L.Conv2D(num_filters,1,padding="same")(g)
    Wg=L.BatchNormalization()(Wg)
    Ws=L.Conv2D(num_filters,1,padding="same")(s)
    Ws=L.BatchNormalization()(Ws)

    out = L.Activation("relu")(Wg + Ws)
    out = L.Conv2D(num_filters, 1, padding="same")(out)
```

```

    out = L.Activation("sigmoid")(out)

    return out * s

def encoder_block(x, num_filters):
    x = conv_block(x, num_filters)
    p = L.MaxPool2D((2, 2))(x)
    return x, p

# Convolution Block: Used in both encoder and decoder.

def conv_block(input, num_filters):

    x = Conv2D(num_filters, 3, padding="same")(input)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)

    x = Conv2D(num_filters, 3, padding="same")(x)

```

```

    x = BatchNormalization()(x)
    x = Activation("relu")(x)

    return x

# Decoder Block

def decoder_block(x, s, num_filters, attention=True):
    x = Conv2DTranspose(num_filters, (2, 2), strides=2, padding="same")(x)
    s = attention_gate(x, s, num_filters)
    x = Concatenate()([x, s])
    x = conv_block(x, num_filters)
    return x

```

```
def attention_unet(input_shape):
    inputs = L.Input(input_shape)
    #Inp: 256, 256, 3 Out: (256, 256, 64), (128, 128, 64)
    s1, p1 = encoder_block(inputs, 64)
    #Inp: 128, 128, 64 Out: (128, 128, 128), (64, 64, 128)
    s2, p2 = encoder_block(p1, 128)
    #Inp: 64, 64, 128 Out: (64, 64, 256), (32, 32, 256)
    s3, p3 = encoder_block(p2, 256)
    #Inp: 32, 32, 256 Out: (32, 32, 512)
    b1 = conv_block(p3, 512)
    d1 = decoder_block(b1, s3, 256)
    d2 = decoder_block(d1, s2, 128)
    d3 = decoder_block(d2, s1, 64)
    outputs = L.Conv2D(1, 1, padding="same", activation="sigmoid")(d3)
    model = Model(inputs, outputs, name="Attention-UNET")
    return model
```

```
input_shape = (256, 256, 3)
model = attention_unet(input_shape)
model.summary()
```

```
=====
=====
Total params: 7971585 (30.41 MB)
Trainable params: 7964161 (30.38 MB)
Non-trainable params: 7424 (29.00 KB)
```

4.6.5.1 L'entraînement du modèle

```
EPOCHS = 100
BATCH_SIZE = 16
learning_rate = 0.001
train_generator_args = dict(rotation_range=0.1,
                            width_shift_range=0.05,
                            height_shift_range=0.05,
                            shear_range=0.05,
                            zoom_range=0.05,
                            horizontal_flip=True,
                            vertical_flip=True,
                            fill_mode='nearest')

train_generator = image_augmentor(MRI_train, BATCH_SIZE,
                                  train_generator_args,
                                  target_size=(256,256))

validation_generator = image_augmentor(MRI_val, BATCH_SIZE,
                                       dict(),
                                       target_size=(256,256))
```

```
opt = Adam(learning_rate=learning_rate, beta_1=0.9, beta_2=0.9, epsilon=1e-07, amsgrad=False)

callbacks = [ModelCheckpoint('MRI_Attention_UNet_ResNet.hdf5', verbose=1, save_best_only=True),
            ReduceLROnPlateau(monitor='val_loss', factor=0.1, verbose=1, min_lr=1e-11)]

model.compile(optimizer=opt, loss=dice_loss, metrics=[iou, dice_coef, Precision(), Recall()])
```

```

history = model.fit(train_generator,
                     steps_per_epoch=len(MRI_train) // 16,
                     epochs=EPOCHS,
                     callbacks=callbacks,
                     validation_data = validation_generator,
                     validation_steps=len(MRI_val) // 16)

```

Pendant la phase d'entraînement du modèle Attention UNet sur 100 époques, chaque itération complète à travers l'ensemble d'entraînement est caractérisée par un lot de 16 images, où une diversité de transformations, telles que la rotation, le décalage, et les retournements, est appliquée pour augmenter la variabilité des données. L'optimiseur Adam, configuré avec un taux d'apprentissage de 0.001, supervise l'ajustement des poids du modèle. Les générateurs d'images utilisent ces augmentations aussi bien pour l'ensemble d'entraînement que de validation. Parallèlement, des mécanismes de rappel, tels que ModelCheckpoint et ReduceLROnPlateau, sont incorporés pour sauvegarder le modèle au meilleur moment et ajuster de manière adaptative le taux d'apprentissage en fonction de la performance sur les données de validation. Cette stratégie d'entraînement permet d'optimiser efficacement le modèle Attention UNet, assurant ainsi une segmentation sémantique précise et robuste pour des applications variées.

4.6.5.2 Les résultats de l'entraînement

Tout au long de l'entraînement du modèle Attention UNet sur 100 époques, des métriques cruciales ont été suivies pour évaluer la performance et l'adaptabilité du modèle. Lors de l'époque 100, la méthode ReduceLROnPlateau a réduit le taux d'apprentissage à 1.0000001111620805e-07, démontrant une réactivité du modèle aux variations de complexité dans les données. À cette étape, le modèle a atteint une perte minimale de 0.1238 sur l'ensemble d'entraînement, tandis que les mesures d'évaluation telles que l'indice Jaccard (IOU) de 0.7889, le coefficient Dice de 0.8762, la précision de 0.9043, et le rappel de 0.8691 soulignent la capacité du modèle à maintenir des performances élevées. La phase de validation a également maintenu des résultats robustes, avec une perte de 0.1116, un IOU de 0.8036, un Dice de 0.8890, une précision de 0.9072, et un rappel de 0.8777. Ces observations globales mettent en lumière la résilience du modèle dans l'ajustement continu de son apprentissage, renforçant ainsi sa capacité à effectuer une segmentation sémantique précise et adaptable sur des données médicales complexes.

```
Epoch 1/100
221/221 [=====] - ETA: 0s - loss: 0.6772 - iou: 0.2124 - dice_coef: 0.3228 - precision_1: 0.2090 - recall_1: 0.6487
Found 197 validated image filenames.
Found 197 validated image filenames.

Epoch 1: val_loss improved from inf to 0.94531, saving model to MRI_Attention_UNet_ResNet.hdf5
221/221 [=====] - 132s 528ms/step - loss: 0.6772 - iou: 0.2124 - dice_coef: 0.3228 - precision_1: 0.2090 - recall_1: 0.6487 - val_loss: 0.9453 - val_iou: 0.0283 - val_dice_coef: 0.0547 - val_precision_1: 0.0291 - val_recall_1: 0.9982 - lr: 0.0010
Epoch 2/100
221/221 [=====] - ETA: 0s - loss: 0.4371 - iou: 0.4142 - dice_coef: 0.5629 - precision_1: 0.5763 - recall_1: 0.5785
Epoch 2: val_loss improved from 0.94531 to 0.89249, saving model to MRI_Attention_UNet_ResNet.hdf5

Epoch 99: val_loss did not improve from 0.07992
221/221 [=====] - 115s 519ms/step - loss: 0.1270 - iou: 0.7850 - dice_coef: 0.8730 - precision_1: 0.9000 - recall_1: 0.8670 - val_loss: 0.1262 - val_iou: 0.7846 - val_dice_coef: 0.8769 - val_precision_1: 0.9040 - val_recall_1: 0.8441 - lr: 1.0000e-06
Epoch 100/100
221/221 [=====] - ETA: 0s - loss: 0.1238 - iou: 0.7889 - dice_coef: 0.8762 - precision_1: 0.9043 - recall_1: 0.8691
Epoch 100: val_loss did not improve from 0.07992

Epoch 100: ReduceLROnPlateau reducing learning rate to 1.0000001111620805e-07.
221/221 [=====] - 116s 524ms/step - loss: 0.1238 - iou: 0.7889 - dice_coef: 0.8762 - precision_1: 0.9043 - recall_1: 0.8691 - val_loss: 0.1116 - val_iou: 0.8036 - val_dice_coef: 0.8890 - val_precision_1: 0.9072 - val_recall_1: 0.8777 - lr: 1.0000e-06
```

Le graphique ci-dessous présente l'évolution des métriques de performance sur les 100 époques d'entraînement du modèle Attention Unet.

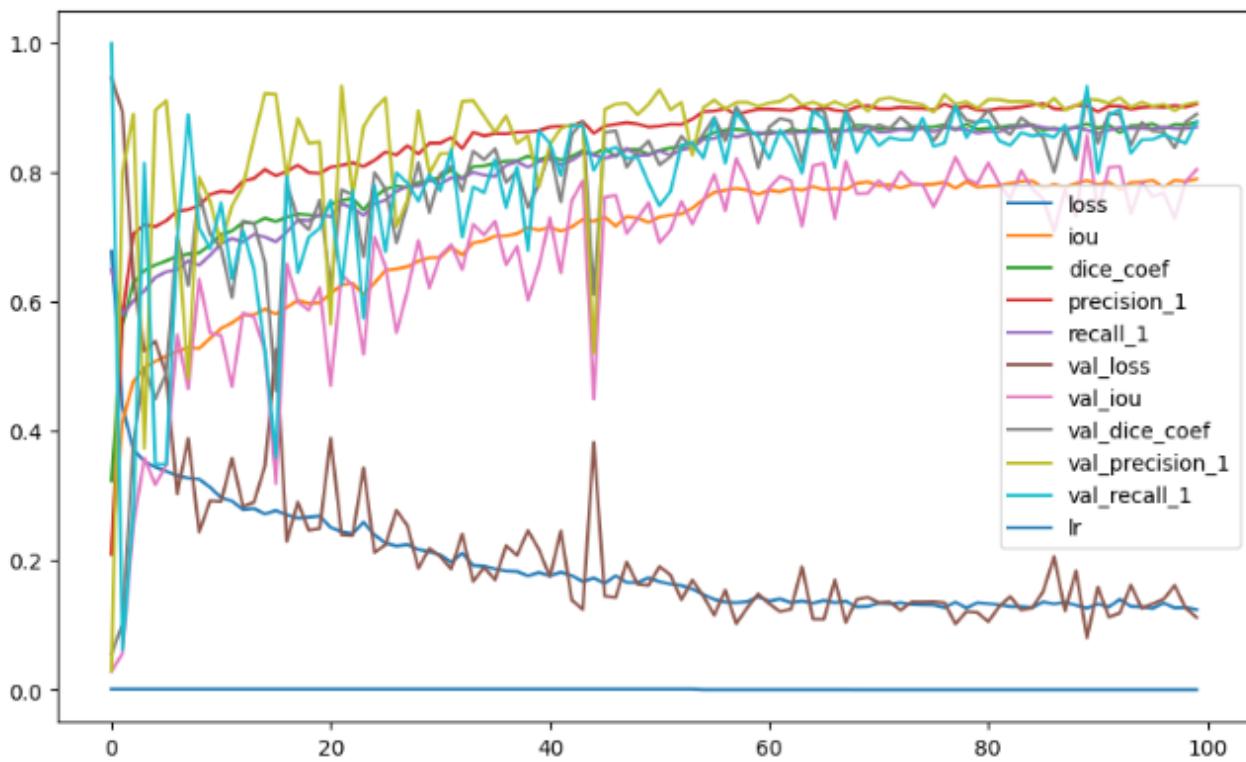


FIGURE 4.10: Synthèse de la performance du modèle Attention Unet : Données d'entraînement et de validation

4.6.5.3 L'évaluation du modèle

L'évaluation du modèle sur les données de test révèle des performances solides et prometteuses. La perte obtenue est de 0.1155, démontrant une faible déviation entre les prédictions du modèle et les véritables annotations. L'indice Jaccard (IOU) de 0.8084 indique un bon chevauchement entre les masques prédits et les masques de référence, témoignant d'une précision élevée dans la délimitation des structures sémantiques. Le coefficient Dice de 0.8895, mesurant la similarité entre les prédictions et les véritables labels, renforce cette précision. La précision de 0.9196 souligne le pourcentage de pixels correctement identifiés par le modèle parmi ceux qu'il a prédits comme positifs. Enfin, le rappel de 0.8819 indique la capacité du modèle à capturer la majorité des pixels réellement positifs. Ces résultats suggèrent que le modèle est capable d'effectuer une segmentation sémantique précise sur les données de test.

Pour l'entraînement du modèle UNet, nous avons utilisé un optimiseur Adam avec un taux d'apprentissage de 0.001. La fonction de perte choisie était Dice.

Le modèle a été entraîné sur 100 époques, avec des évaluations périodiques de la performance sur

```

test_gen = image_augmentor(MRI_test, BATCH_SIZE,
                           dict(),
                           target_size=(256,256))
results = model.evaluate(test_gen, steps=len(MRI_test) / BATCH_SIZE)
print("Test IOU: ", results[1])
print("Test Dice Coefficient: ", results[2])

```

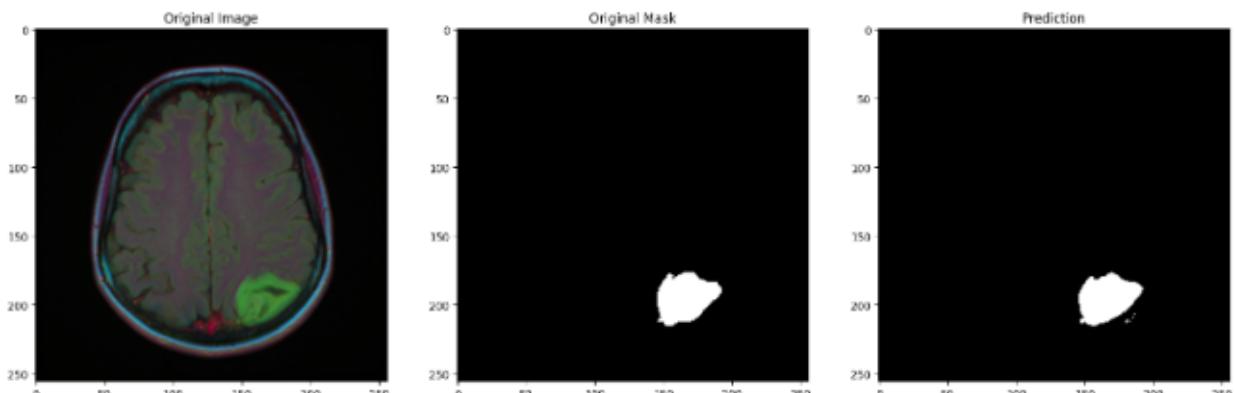
```

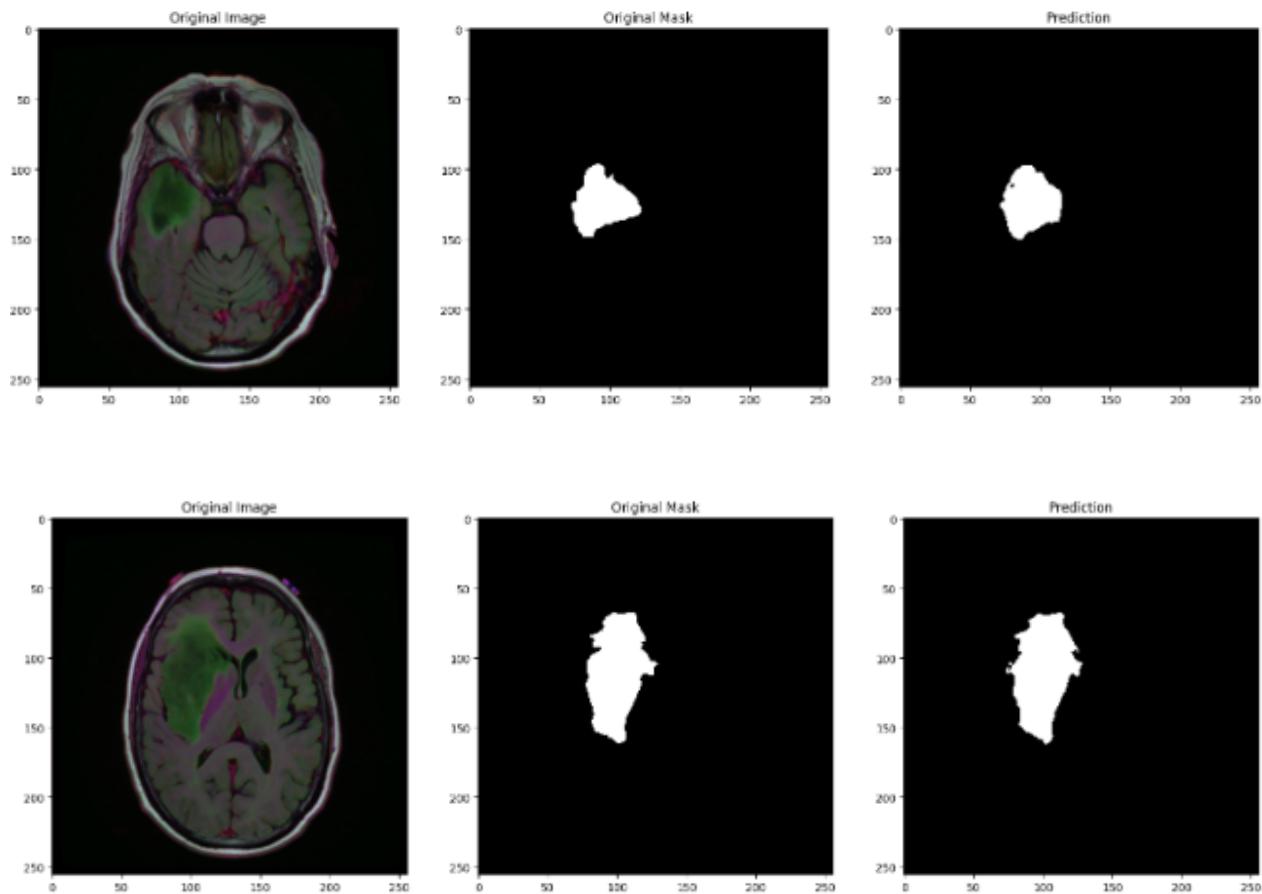
12/12 [=====] - 4s 315ms/step - loss: 0.1155 -
iou: 0.8084 - dice_coef: 0.8895 - precision_1: 0.9196 - recall_1: 0.8819
Test IOU:  0.8084484934806824
Test Dice Coefficient:  0.8894612193107605

```

un ensemble de validation. Un mécanisme d'arrêt prématué (early stopping) a été mis en place pour terminer l'entraînement si la performance de validation cesse de s'améliorer pendant un nombre consécutif d'itérations.

4.6.5.4 Tester le modèle sur les images du test





4.7 Comparaison entre les modèles

Modèles	IOU	Dice Coef	Recall	Precision	Loss
DeeppLabV3+	0.7907	0.8824	0.9066	0.8982	0.1176
PSPNet	0.6393	0.7732	0.7954	0.8086	0.2268
SegNet	0.7323	0.8435	0.8902	0.8269	0.1565
Unet	0.7217	0.8396	0.8234	0.910	0.1337
Attention-Unet	0.8084	0.8895	0.8819	0.9196	0.1155

TABLE 4.1: Comparaison des Performances de Modèles de Segmentation avec leurs Mesures d’Évaluation et Fonctions de Coût Associées

4.8 Conclusion

En conclusion, après avoir évalué les performances des modèles DeeplabV3+, PSPNet, SegNet, Unet et Attention-Unet sur différentes métriques telles que l’IOU, le Dice Coefficient, le Recall, la Precision et la Loss, il est clair que DeeplabV3+ et Attention-Unet émergent comme les choix les plus robustes avec des scores élevés dans toutes les catégories. Unet et SegNet présentent également des performances solides, tandis que PSPNet affiche des résultats légèrement inférieurs.

5

Conclusion

Notre étude a pris en considération une variété d'architectures de modèles de segmentation, à savoir SegNet, Attention UNet, UNet, DeepLabV3+, et PSPNet, pour la détection des tumeurs cérébrales. Chacune de ces architectures a été minutieusement choisie pour ses attributs uniques dans le contexte spécifique de l'imagerie médicale.

SegNet, avec son architecture encodeur-décodeur, a montré son efficacité à préserver les détails spatiaux tout en capturant des informations sémantiques cruciales. L'ajout du mécanisme d'attention dans le modèle Attention UNet a permis une focalisation accrue sur des régions spécifiques, conduisant à des améliorations significatives. L'UNet classique a confirmé son rôle robuste dans la segmentation d'images médicales avec une performance stable.

Les modèles plus avancés, tels que DeepLabV3+ et PSPNet, ont introduit des approches novatrices. DeepLabV3+ a utilisé des convolutions atrous pour capturer des détails fins sans compromettre la qualité de la segmentation. D'un autre côté, PSPNet a exploité des modules de mise à l'échelle spatiale pour obtenir une compréhension holistique des structures à différentes échelles.

Cette diversité d'approches nous a permis d'obtenir des perspectives complètes sur les performances et les spécificités de chaque modèle dans le contexte de la détection des tumeurs cérébrales. En évaluant ces architectures sous diverses conditions, notre projet a contribué à une compréhension approfondie de leurs avantages respectifs, ouvrant la voie à des choix d'architecture plus informés dans des scénarios cliniques spécifiques. En combinant ces différents modèles, notre approche s'inscrit dans une démarche holistique visant à améliorer continuellement les capacités de détection et de caractérisation des tumeurs cérébrales par le biais de l'apprentissage en profondeur.

À l'issue de ce projet dédié à l'application des modèles de segmentation pour la détection des tumeurs cérébrales, plusieurs perspectives émergent pour guider les développements futurs. La première consiste à explorer l'intégration de données multi-modales afin d'élargir la vision diagnostique. L'optimisation continue des hyperparamètres demeure une piste cruciale pour maximiser les

performances spécifiques de chaque modèle. L'utilisation d'ensemble learning pourrait renforcer la robustesse de l'approche en combinant les forces de plusieurs modèles. L'accent sur l'interprétabilité des modèles et leur adaptation à de nouvelles données garantirait une utilisation clinique plus sûre et évolutive. L'extension de l'application des modèles à d'autres pathologies cérébrales et leur déploiement clinique requièrent une approche holistique et une collaboration étroite avec les professionnels de la santé. En résumé, ces perspectives tracent la voie vers une utilisation plus avancée et adaptative des modèles de segmentation, offrant ainsi des perspectives prometteuses pour l'amélioration continue des pratiques diagnostiques en neurologie.

Bibliographie

- [1] Architecture FCN : <https://arxiv.org/abs/1411.4038>. (s. d.).
- [2] BADRINARAYANAN V, C. R., Kendall A. (2017). Segnet : a deep convolutional encoder–decoder architecture for image segmentation. *IEEE Trans Pattern Anal Mach Intell*.
- [3] BUDA, M. (2019). LGg MRI Segmentation Dataset."<https://www.kaggle.com/mateuszbuda/lgg-mri-segmentation>.
- [4] Convolution atreuse : <https://arxiv.org/abs/1606.00915>. (s. d.).
- [5] Convolution séparable : <https://arxiv.org/abs/1802.02611>. (s. d.).
- [6] GHANEI, H. S.-Z., & WINDHAM., J. (1998). Segmentation of the hippocampus from brain MRI using deformable contours. *Computerized Medical Imaging and Graphics*,
- [7] GORDILLO N, S. P., Montseny E. (2013). State of the art survey on MRI brain tumor segmentation.
- [8] J.C. BEZDEK, L. H., & CLARKE., L. (1993). Review of MR image segmentation techniques using pattern recognition. *Medical Physics*.
- [9] MARAIS & BRADY., J. (2000). Detecting the brain surface in sparse MRI using boundary models. *Medical Image Analysis*.
- [10] MARK H. BILSKY, W. M. C. o. C. U., MD. (mai 2023). Présentation des tumeurs cérébrales.
- [11] Modèle Deeplabv3+ : <https://arxiv.org/abs/1802.02611>. (s. d.).
- [12] Organigramme Deeplabv1 : <https://arxiv.org/abs/1606.00915>. (s. d.).
- [13] Pooling de pyramides spatiales Atrous (ASPP) : <https://arxiv.org/abs/1606.00915>. (s. d.).
- [14] RONNEBERGER O, B. T., Fischer P. (2015). U-net : convolutional networks for biomedical image segmentation. In : International conference on medical image computing and computer-assisted intervention1.
- [15] SCHÜRMANN., J. (1996). Pattern classification. A unified view of statistical and neural approaches.

- [16] Segmentation en Contours,<https://patrick-bonnin.developpez.com/cours/vision/apprendre-bases-traitement-image/>. (s. d.).
- [17] Segmentation en régions,<https://xphilipp.developpez.com/articles/segmentation/regions/>. (s. d.).
- [18] TECHNICAL REPORT, É. d. M. d. P. (1997). C. Ambroise. Introduction à la reconnaissance statistique des formes.