ÉCOLE NATIONALE SUPÉRIEURE DES TECHNIQUES AVANCÉES
ÉCOLE NATIONALE D'INGÉNIEURS DE TUNIS

SIM 202

Image Segmentation

Réalisé par :
Hana FEKI
Rayen ZARGUI
Rayen MANSOUR

Classe :
2ème année Mathématiques appliquées

Encadré par :
Mme Sonia ALOUANE

Année universitaire 2024/2025

Table des matières

Table des figures

Introduction

La vision par ordinateur, en tant que domaine interdisciplinaire combinant l'informatique, les mathématiques et l'intelligence artificielle, vise à permettre aux machines d'interpréter et d'analyser visuellement le monde, à l'image de la perception humaine. Parmi les défis fondamentaux de cette discipline, **la segmentation d'images occupe une place centrale**, car elle consiste à partitionner une image en régions significatives pour faciliter son analyse, son traitement ou son utilisation dans des applications variées, telles que la médecine, la robotique ou l'analyse d'images industrielles.

Ce projet s'inscrit dans cette problématique en explorant et en implémentant des algorithmes de segmentation, avec **un focus initial sur la méthode de Chan-Vese**, avant d'envisager d'autres approches comme la méthode d'Otsu pour améliorer les résultats.

Nous avons également élargi notre recherche à des applications spécifiques, comme **la détection de tumeurs cérébrales en 3D à partir d'IRM**, où l'utilisation de Python et des techniques d'apprentissage profond a permis d'obtenir des résultats encourageants.

Ce rapport détaille les étapes de conception, d'implémentation, de validation et d'optimisation, tout en identifiant les défis rencontrés et les perspectives d'amélioration pour des solutions plus robustes et performantes.

Chapitre 1

Contexte

Dans le cadre de ce projet, l'algorithme de Chan-Vese a été initialement exploré comme une méthode pour la segmentation d'images, en raison de sa capacité à segmenter des régions homogènes tout en respectant les frontières des objets. Cependant, après avoir appliqué cet algorithme à notre ensemble de données, nous avons constaté que **l'algorithme de Chan-Vese seul ne donnait pas des résultats satisfaisants**, notamment en termes de précision des contours détectés et de robustesse face au bruit ou aux contours flous. La segmentation obtenue était parfois imprécise, avec des contours mal définis ou des objets mal segmentés.

Face à ces limitations, nous avons décidé d'affiner l'algorithme de Chan-Vese en y intégrant des techniques de prétraitement, comme l'utilisation de filtres pour réduire le bruit dans les images avant la segmentation. Ces filtres, tels que le filtre gaussien, permettent de **lisser l'image et d'améliorer la détection des contours**, tout en minimisant l'impact du bruit sur la performance de l'algorithme. Cela a permis d'améliorer significativement la qualité des contours extraits et d'obtenir des résultats plus fiables dans notre contexte d'application.

1.1 Analyse des besoins et faisabilité

1.1.1 Analyse des besoins

Le besoin principal est de détecter de manière efficace et précise les contours des objets dans des images afin de faciliter des tâches de reconnaissance, d'analyse et de classification. Cette segmentation des contours est cruciale pour des applications telles que la reconnaissance d'objets, la détection de formes, ou encore l'amélioration des performances des modèles de vision par ordinateur. Une segmentation précise des contours permet de mieux délimiter les objets d'intérêt et de réduire les erreurs dans les étapes suivantes du traitement de l'image.

1.1.2 Faisabilité

L'algorithme de Chan-Vese est basé sur une minimisation d'une énergie fonctionnelle qui permet de segmenter des régions homogènes tout en respectant les contours. Cependant, son efficacité peut être affectée par des facteurs comme le bruit ou des

objets mal délimités dans l'image. Pour améliorer la robustesse et la précision de l'algorithme, nous avons combiné l'approche de Chan-Vese avec des filtres de prétraitement, comme des filtres de lissage (par exemple, le filtre gaussien), afin de réduire le bruit et de rendre les contours plus nets. Cela a permis d'améliorer les performances de la segmentation tout en préservant les détails importants de l'image.

1.2 Spécification

1.2.1 Entrées

Le système reçoit en entrée des images, qu'elles soient en couleur ou en niveaux de gris. Ces images, provenant de diverses sources, sont souvent affectées par du bruit ou un faible contraste. Un prétraitement s'avère donc nécessaire pour réduire le bruit et améliorer la netteté des contours détectés. Ce prétraitement inclut des techniques de lissage, comme l'utilisation d'un filtre gaussien, ainsi que la normalisation de l'image.

1.2.2 Sortie

La sortie du système est une image binaire où les contours des objets sont clairement détectés. Les contours sont délimités avec plus de précision après l'application des filtres et de l'algorithme de Chan-Vese affiné, permettant ainsi une meilleure analyse des objets présents dans l'image.

Chapitre 2

Conception

2.1 Conception architecturale

2.1.1 Diagramme de classes

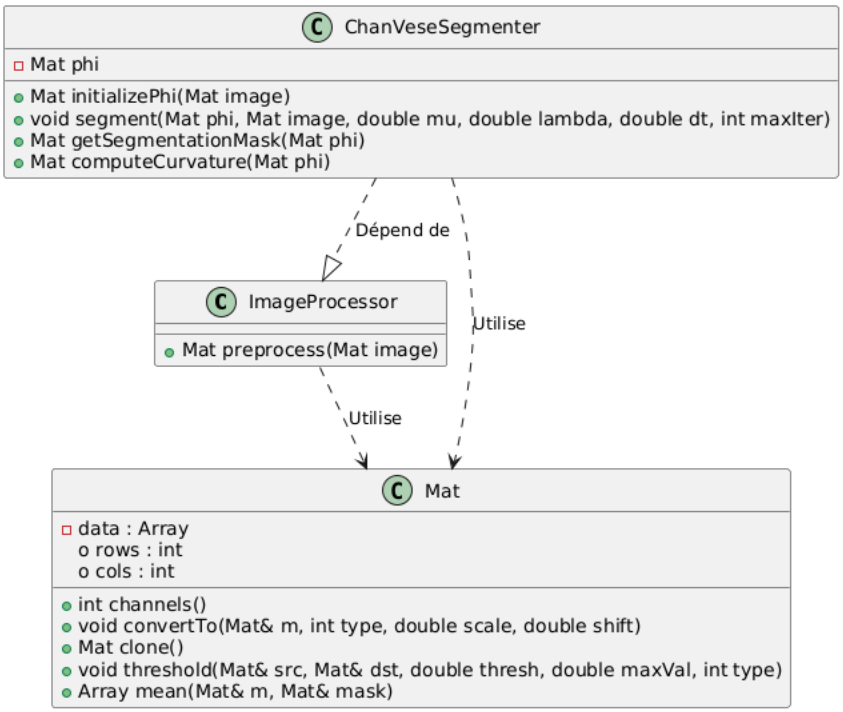

FIGURE 2.1 – Diagramme de classes

2.1.2 Interaction entre les classes

Le diagramme de classe illustre l'interaction entre trois classes principales : `ImageProcessor`, `ChanVeseSegmenter`, et `Mat`.

— **ImageProcessor** : La classe ImageProcessor est responsable du prétraitement de l'image. Elle effectue des opérations telles que la conversion en niveaux de gris, l'application d'un flou gaussien et l'égalisation de l'histo-

gramme. Une fois l'image prétraitée, elle est transmise à la classe ChanVese-Segmenter pour la segmentation.

— **ChanVeseSegmenter** : Cette classe applique l'algorithme de segmentation de Chan-Vese sur l'image prétraitée afin de séparer les régions d'intérêt. Elle utilise également des méthodes pour initialiser la fonction phi, effectuer l'évolution des contours et générer un masque binaire représentant les régions segmentées.

— **Mat** : Les deux classes, ImageProcessor et ChanVeseSegmenter, utilisent la classe Mat de la bibliothèque OpenCV pour manipuler les données d'image sous forme de matrices. Ainsi, elles interagissent principalement à travers des objets Mat, ce qui permet de réaliser efficacement les différentes étapes du traitement et de la segmentation de l'image.

2.2 Conception détaillée

2.2.1 Librairies utilisées

Ce projet utilise la bibliothèque OpenCV pour le traitement d'images. OpenCV permet d'effectuer des opérations classiques telles que la conversion en niveaux de gris, l'application de flou gaussien et l'égalisation d'histogramme. Elle propose également des fonctionnalités avancées pour la segmentation d'images et une interface pratique pour manipuler des images sous forme de matrices avec la classe `Mat`.

2.2.2 Classes : attributs et méthodes

Classe `ImageProcessor`

La classe `ImageProcessor` est responsable du prétraitement de l'image avant la segmentation. Elle possède la méthode suivante :

— `preprocess(const Mat &image)` : Prend une image en entrée. Si l'image est en couleur, elle la convertit en niveaux de gris. Elle applique ensuite un flou gaussien pour lisser l'image, égalise l'histogramme pour améliorer le contraste, et convertit l'image en une matrice de type `CV_64F` pour la normaliser entre 0 et 1. La méthode retourne l'image prétraitée.
Ces étapes rendent l'image plus adaptée à l'algorithme de segmentation en réduisant le bruit et en améliorant le contraste des régions d'intérêt.

Classe `ChanVeseSegmenter`

La classe `ChanVeseSegmenter` applique l'algorithme de Chan-Vese pour la segmentation d'image. Elle possède les méthodes suivantes :

— `initializePhi(const Mat &image)` : Initialise la fonction phi en créant un disque centré au milieu de l'image. Les pixels à l'intérieur du disque sont définis comme étant à -1 et ceux à l'extérieur à +1, ce qui permet de définir une frontière initiale pour la segmentation.

— `segment(Mat &phi, const Mat &image, double mu, double lambda, double dt, int maxIter)` : Effectue la segmentation en utilisant l'algorithme de

Chan-Vese. L'algorithme utilise une approche itérative pour ajuster phi en fonction des forces de données (différences d'intensité) et des forces de courbure. La segmentation s'arrête après un nombre d'itérations fixé.

— `getSegmentationMask(const Mat &phi)` : Génère un masque binaire à partir de la fonction `phi`. Les pixels à l'intérieur du contour sont définis à 255 (blanc), tandis que les pixels à l'extérieur sont définis à 0 (noir).

— `computeCurvature(const Mat &phi)` : Détecte et dessine les contours trouvés sur l'image d'origine. Seuls les contours ayant une aire suffisante (supérieure à un seuil défini) sont dessinés pour éviter le bruit. L'image avec les contours dessinés est ensuite enregistrée sous le nom de fichier spécifié.

Ces classes travaillent ensemble pour effectuer une segmentation d'image basée sur la méthode de Chan-Vese. La classe ImageProcessor prépare l'image, tandis que la classe ChanVeseSegmenter effectue la segmentation proprement dite.

2.2.3 Principes de fonctionnement

Le processus commence par le prétraitement de l'image avec la classe ImageProcessor, qui rend l'image plus adaptée à l'algorithme de segmentation. Ensuite, l'image prétraitée est passée à la classe ChanVeseSegmenter, qui initialise une fonction phi représentant le contour de la région à segmenter. L'algorithme de Chan-Vese est itéré, ajustant phi jusqu'à ce que le contour converge pour séparer la région d'intérêt de l'arrière-plan. Enfin, un masque binaire est généré et enregistré comme résultat de la segmentation sous forme d'image PNG.

Chapitre 3

Implémentation

3.1 Algorithme de Chan-Vese

L'algorithme de Chan–Vese est une méthode de segmentation d'image basée sur un modèle variationnel. Contrairement aux méthodes fondées sur le calcul de gradients, le modèle de Chan–Vese repose sur l'hypothèse que l'image se compose de deux régions relativement homogènes (l'intérieur et l'extérieur d'un contour). Le but est de déterminer un contour évolutif qui sépare ces deux régions en minimisant un fonctionnel d'énergie. Cette approche utilise une fonction de niveau pour représenter implicitement le contour, ce qui permet une évolution continue de la frontière.

3.1.1 Modélisation Mathématique

Soit une image $I(x, y)$ définie sur un domaine $\Omega \subset \mathbb{R}^2$. On souhaite segmenter l'image en deux régions distinctes : l'intérieur et l'extérieur d'un contour C. Le modèle de Chan–Vese définit le fonctionnel d'énergie suivant :

$$E(c_1, c_2, C) = \mu \,\mathrm{Length}(C) + \lambda_1 \int_{inside(C)} |I(x,y) - c_1|^2 \, dx\, dy + \lambda_2 \int_{outside(C)} |I(x,y) - c_2|^2 \, dx\, dy,$$

où c_1 et c_2 représentent les moyennes d'intensité à l'intérieur et à l'extérieur du contour C, respectivement, et μ, λ_1 et λ_2 sont des paramètres qui pondèrent l'importance de chaque terme.

Pour faciliter l'implémentation, le contour C est représenté par une fonction de niveau $\phi(x, y)$ telle que :

$$C = \{(x, y) \in \Omega \mid \phi(x, y) = 0\}.$$

Les régions sont alors définies par :

$$\text{Intérieur} : \phi(x, y) < 0, \quad \text{Extérieur} : \phi(x, y) \geq 0.$$

Le fonctionnel peut être réécrit en termes de ϕ en utilisant la fonction de Heaviside

$H(\phi)$ et sa dérivée $\delta(\phi)$ (une approximation de la fonction delta de Dirac) :

$$\begin{aligned} E(\phi, c_1, c_2) = \mu \int_\Omega \delta(\phi(x,y)) \, |\nabla \phi(x,y)| \, dx \, dy \\ + \lambda_1 \int_\Omega H(\phi(x,y)) \, |I(x,y) - c_1|^2 \, dx \, dy \\ + \lambda_2 \int_\Omega \Big(1 - H(\phi(x,y))\Big) \, |I(x,y) - c_2|^2 \, dx \, dy. \end{aligned}$$

L'objectif est de faire évoluer $\phi(x, y)$ de manière à minimiser $E(\phi, c_1, c_2)$. La minimisation se fait typiquement par descente de gradient, conduisant à l'équation d'évolution suivante :

$$\frac{\partial \phi}{\partial t} = \delta(\phi) \left[\mu \, \mathrm{div} \left(\frac{\nabla \phi}{|\nabla \phi|} \right) - \lambda_1 \left(I - c_1 \right)^2 + \lambda_2 \left(I - c_2 \right)^2 \right].$$

Cette équation permet de mettre à jour la fonction de niveau ϕ à chaque itération.

3.1.2 Pseudocode de l'Algorithme de Chan–Vese

Le processus algorithmiquement se déroule de la manière suivante :

Soit une image $I(x, y)$ et une fonction de niveau initiale $\phi(x, y)$ obtenue par un seuillage ou une autre méthode d'initialisation. L'algorithme itératif alterne entre le calcul des moyennes régionales et l'évolution de ϕ.

Initialisation : Définir $\phi(x, y)$ pour $(x, y) \in \Omega$.

Itérations : Répéter jusqu'à convergence :

1. Calculer les moyennes d'intensité à l'intérieur et à l'extérieur du contour :

$$c_1 = \frac{\int_\Omega I(x,y) H(\phi(x,y)) \, dx \, dy}{\int_\Omega H(\phi(x,y)) \, dx \, dy}, \quad c_2 = \frac{\int_\Omega I(x,y) \left[1 - H(\phi(x,y))\right] \, dx \, dy}{\int_\Omega \left[1 - H(\phi(x,y))\right] \, dx \, dy}.$$

2. Évoluer $\phi(x, y)$ par la descente de gradient de l'énergie :

$$\frac{\partial \phi}{\partial t} = \delta(\phi) \left[\mu \, \mathrm{div} \left(\frac{\nabla \phi}{|\nabla \phi|} \right) - \lambda_1 \left(I - c_1 \right)^2 + \lambda_2 \left(I - c_2 \right)^2 \right].$$

3. Réinitialiser $\phi(x, y)$ si nécessaire pour assurer la stabilité numérique.

Le processus se poursuit jusqu'à ce que la modification de $\phi(x, y)$ soit infime, indiquant la convergence du contour $C = \{\phi(x, y) = 0\}$.

3.2 Prétraitement de l'image

Avant d'appliquer l'algorithme de **Chan-Vese**, il est crucial de préparer l'image à travers différentes étapes de prétraitement. Ces étapes visent principalement à réduire le bruit, à améliorer le contraste et à faciliter l'identification des contours. En effet, un bon prétraitement garantit non seulement une meilleure convergence de l'algorithme, mais aussi une segmentation plus précise.

3.2.1 Conversion en niveaux de gris

La première étape consiste à convertir l'image en niveaux de gris. Cette transformation est essentielle car l'algorithme de Chan-Vese fonctionne principalement sur des variations d'intensité, et non sur des informations de couleur.

Si l'image d'origine comporte trois canaux de couleur (RVB), elle est convertie à l'aide de la transformation suivante :

$$I_{\text{gris}}(x, y) = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$

Cette conversion permet de simplifier les données et de se concentrer uniquement sur les variations d'intensité lumineuse.

```
// Conversion en niveaux de gris
    if (image.channels() == 3)
        cvtColor(image, gray, COLOR_BGR2GRAY);
    else
        gray = image.clone();
}
```

3.2.2 Réduction du bruit par filtrage gaussien

Les images réelles contiennent souvent du bruit, qui peut fortement perturber le processus de segmentation. Pour pallier ce problème, on utilise un **filtre gaussien**. Ce filtre applique un lissage qui permet d'atténuer le bruit tout en préservant les structures importantes de l'image.

Le filtre est défini par la convolution de l'image avec une fonction gaussienne :

$$G(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

Dans notre cas, nous utilisons un noyau de taille 5×5 avec un écart-type (σ) de 1.0. Ce lissage permet de réduire les variations d'intensité trop abruptes dues au bruit, sans altérer les contours significatifs.

```
// Filtrage gaussien pour r duire le bruit
GaussianBlur(gray, blurred, Size(5, 5), 1.0);
```

3.2.3 Amélioration du contraste par égalisation d'histogramme

Après le lissage, l'étape suivante consiste à améliorer le contraste de l'image à l'aide de l'**égalisation d'histogramme**. Cette technique redistribue les niveaux de gris pour utiliser toute la plage d'intensité disponible (de 0 à 255), augmentant ainsi la distinction entre différentes régions de l'image.

L'équation de transformation de l'intensité est la suivante :

$$I'(x, y) = \text{round}\left(\frac{255}{N} \sum_{k=0}^{I(x,y)} p_k\right)$$

où N est le nombre total de pixels et p_k la probabilité d'apparition du niveau d'intensité k.

Cette transformation permet de révéler des détails dans les zones sombres ou trop claires de l'image.

```
// Amelioration du contraste par egalisation d'histogramme
equalizeHist(blurred, equalized);
```

Enfin, l'image est normalisée pour que les valeurs soient comprises entre 0 et 1, facilitant ainsi son traitement ultérieur.

3.3 Conclusion

L'ensemble des étapes de prétraitement permet d'améliorer la qualité de l'image avant l'application de l'algorithme de Chan-Vese. La conversion en niveaux de gris, le filtrage gaussien et l'égalisation d'histogramme contribuent à optimiser la précision de la segmentation en minimisant les perturbations liées au bruit et aux variations d'intensité.

Chapitre 4

Tests de validation

Les tests de validation sont essentiels pour garantir la robustesse, la fiabilité et la précision de l'implémentation de l'algorithme de segmentation Chan-Vese. Dans ce chapitre, nous détaillons les différents tests réalisés pour vérifier le bon fonctionnement des classes fondamentales de l'application, en abordant à la fois les tests unitaires et les tests de bout en bout.

4.1 Tests Unitaires des Classes Fondamentales

Les tests unitaires visent à valider le comportement des méthodes essentielles de chaque classe : `ImageProcessor` et `ChanVeseSegmenter`.

4.1.1 Classe ImageProcessor

Cette classe s'occupe du prétraitement de l'image : conversion en niveaux de gris, flou gaussien et égalisation d'histogramme.

Objectifs des Tests

- Vérifier la conversion correcte en niveaux de gris.
- Confirmer l'effet du flou gaussien.
- Valider l'égalisation de l'histogramme.

Test de conversion en niveaux de gris

- **Images comparées** :

FIGURE 4.1 – Image Originale

FIGURE 4.2 – Image en niveaux de gris

— **Validation** : Vérification visuelle de l'uniformité des niveaux de gris et conservation des contours principaux.

Test du flou gaussien et égalisation d'histogramme

— **Images comparées** :

FIGURE 4.3 – Image en niveaux de gris

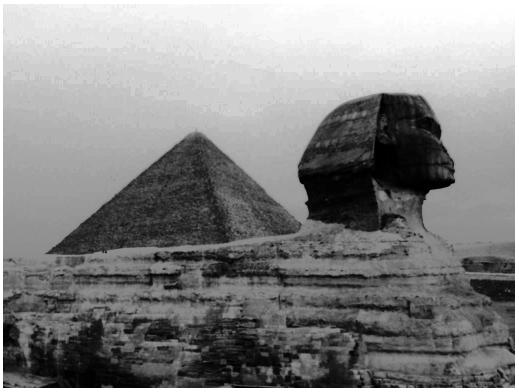

FIGURE 4.4 – Image prétraité

— **Validation** : Les petits détails et le bruit doivent être lissés, tandis que les contours principaux restent visibles.

4.1.2 Classe ChanVeseSegmenter

Cette classe gère l'initialisation du champ de niveau ϕ et la segmentation Chan-Vese.

Objectifs des Tests

— Vérifier l'initialisation correcte de ϕ.
— Observer l'évolution de ϕ au fil des itérations.
— S'assurer de la convergence de l'algorithme.

Tests Réalisés

Test d'initialisation de ϕ

— **Images comparées** :

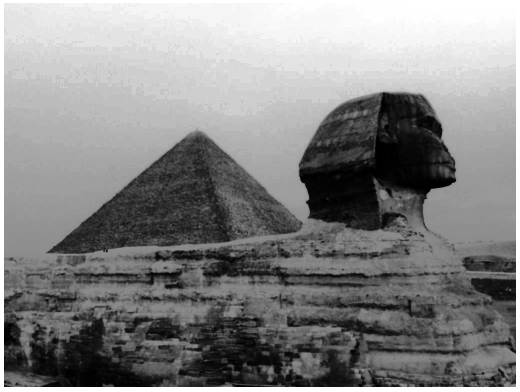

FIGURE 4.5 – Image floue

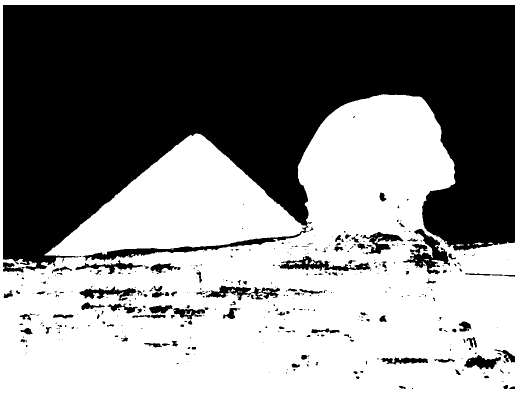

FIGURE 4.6 – Initialisation de ϕ

— **Validation** : Le champ de niveau doit être négatif à l'intérieur des objets et positif à l'extérieur.

— **Résultat attendu** : Les contours initiaux doivent coïncider avec les transitions de luminosité de l'image.

Test d'évolution de ϕ

— **Images comparées** :

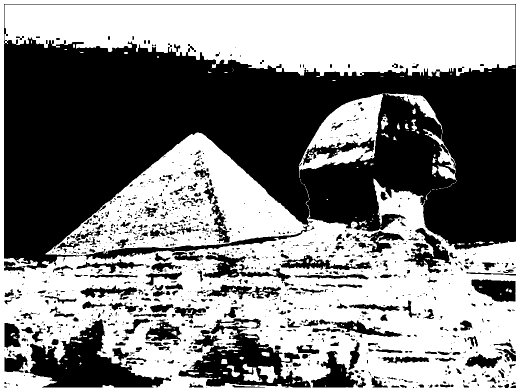

FIGURE 4.7 – ϕ après 50 itérations

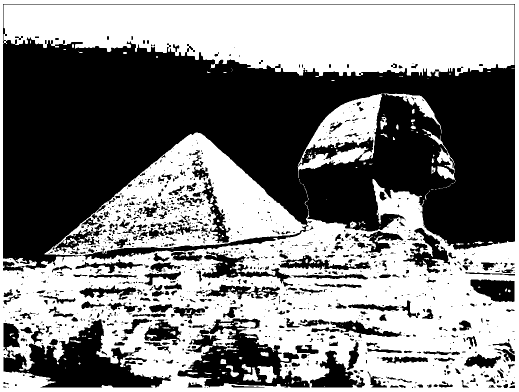

FIGURE 4.8 – ϕ après 100 itérations

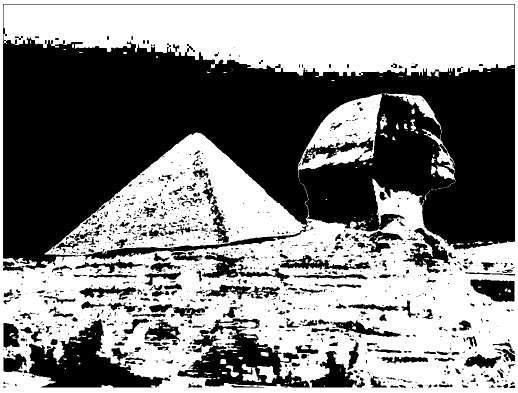

FIGURE 4.9 – ϕ après 150 itérations

— **Validation** : Observation de la progression des contours vers les frontières des objets.

4.2 Tests de Bout en Bout des Classes Fondamentales

4.2.1 Test de Prétraitement et d'Initialisation de ϕ

— **Images comparées** :

FIGURE 4.10 – Image Originale

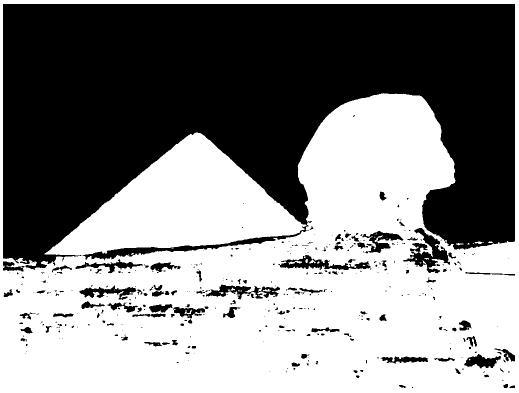

FIGURE 4.11 – Initialisation de ϕ

— **Validation** : Conversion correcte en niveaux de gris et réduction du bruit.
— **Résultat attendu** : ϕ doit refléter les régions intérieures et extérieures des objets.

4.2.2 Test de Segmentation Chan-Vese

— **Images comparées** :

FIGURE 4.12 – image original

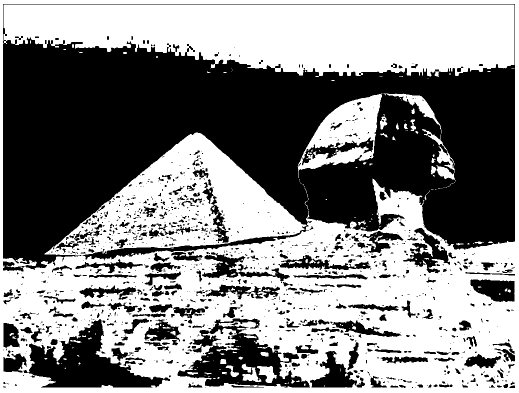

FIGURE 4.13 – Image aprés Segmentation

— **Validation** : Convergence fluide des contours, bien que quelques défauts restent visibles.

4.3 Conclusion des Tests de Validation

Les tests unitaires et de bout en bout confirment le bon fonctionnement des principales étapes de l'algorithme :

— Le **prétraitement** améliore la qualité de l'image pour la segmentation.

— L'**initialisation de** ϕ fournit des contours de départ fiables.

— L'algorithme **Chan-Vese** converge efficacement vers les frontières des objets.

L'algorithme de segmentation Chan-Vese s'est révélé efficace pour détecter les contours des objets dans une image, en particulier lorsque les régions sont bien différenciées en termes d'intensité. Les tests de validation ont démontré que l'initialisation et la convergence de l'algorithme permettent une segmentation globalement fiable.

Cependant, des limites ont été observées. Le coût computationnel élevé peut ralentir le traitement, surtout pour des images de grande taille. De plus, la sensibilité à l'initialisation peut entraîner des erreurs si le contour initial est mal positionné. L'algorithme montre également des difficultés avec les textures complexes ou les images de mauvaise qualité, nécessitant ainsi une prise en charge spécifique pour améliorer la robustesse de la segmentation.

Face à ces limitations, nous avons exploré d'autres méthodes de segmentation afin d'optimiser les performances et d'améliorer la précision des résultats.

Chapitre 5

Exploration d'autres solutions pour la segmentation

5.1 Méthode d'Otsu

La méthode d'Otsu [2] est une technique de seuillage automatique utilisée en traitement d'images pour segmenter une image en deux classes (généralement premier plan et arrière-plan) en se basant sur l'histogramme des niveaux de gris. L'idée principale est de trouver un seuil optimal T qui maximise la *variance inter-classe*. Cela repose sur l'hypothèse que **l'histogramme de l'image est bimodal**.

5.1.1 Explication mathématique

Dans ce rapport, nous allons expliquer chaque étape de cette méthode en partant d'une image initiale et en détaillant le processus jusqu'à obtenir une image binarisée. Prenons l'exemple de l'image suivante, représentée par une grille de niveaux de gris :

$$\begin{pmatrix} 203 & 204 & 205 & 206 & 206 & 206 \\ 198 & 199 & 201 & 191 & 136 & 194 \\ 193 & 166 & 126 & 136 & 61 & 163 \\ 166 & 99 & 97 & 107 & 103 & 164 \\ 109 & 111 & 105 & 107 & 107 & 101 \\ 83 & 84 & 84 & 93 & 100 & 108 \end{pmatrix}$$

Cette image de 6 lignes et 6 colonnes contient 36 pixels, dont les valeurs, appelées intensités, varient entre 61 et 206. Chaque intensité représente un niveau de gris, où 0 correspondrait au noir et 255 au blanc dans une échelle classique de 0 à 255. L'objectif est de trouver un seuil T qui permette de classer chaque pixel soit dans une classe "sombre" (valeur 0), soit dans une classe "claire" (valeur 1). Voici comment la méthode d'Otsu procède, étape par étape.

Étape 1 : Transformer l'image en vecteur aplati

Pour analyser les intensités, on commence par "aplatir" l'image en un seul vecteur contenant toutes les valeurs des pixels, lues ligne par ligne. Cela donne :

$$[203, 204, 205, 206, 206, 206, 198, 199, 201, 191, 136, 194, 193, 166, 126, 136, 61, 163, 166,$$

$$99, 97, 107, 103, 164, 109, 111, 105, 107, 107, 101, 83, 84, 84, 93, 100, 108]$$

Ce vecteur de 36 éléments est plus facile à manipuler pour les calculs qui suivent.

Étape 2 : Calculer l'histogramme et les probabilités

Ensuite, on construit un **histogramme**, qui est une sorte de décompte : pour chaque intensité possible (de 0 à 255), on regarde combien de fois elle apparaît dans le vecteur. Par exemple, la valeur 206 apparaît 3 fois, la valeur 107 apparaît 3 fois également, tandis que la valeur 61 n'apparaît qu'une fois. Voici quelques exemples tirés de notre vecteur :

— Intensité 61 : fréquence = 1
— Intensité 83 : fréquence = 1
— Intensité 84 : fréquence = 2
— Intensité 107 : fréquence = 3
— Intensité 206 : fréquence = 3

Pour les intensités qui n'apparaissent pas (comme 0, 1, ou 255), la fréquence est 0. Cet histogramme nous donne une vue d'ensemble de la répartition des intensités dans l'image.

Puis, on calcule la **probabilité** de chaque intensité. Puisqu'il y a 36 pixels dans l'image, la probabilité P_i d'une intensité i est simplement sa fréquence divisée par le nombre total de pixels :

$$P_i = \frac{\text{Fréquence de l'intensité } i}{36}$$

Par exemple :

— $P_{61} = \frac{1}{36} \approx 0,0278$
— $P_{84} = \frac{2}{36} = \frac{1}{18} \approx 0,0556$
— $P_{107} = \frac{3}{36} = \frac{1}{12} \approx 0,0833$
— $P_{206} = \frac{3}{36} = \frac{1}{12} \approx 0,0833$

Ces moyennes nous indiquent la "valeur typique" de chaque classe pour un seuil donné. Par exemple, si $t = 100$, la classe 1 contient toutes les intensités jusqu'à 100 (comme 61, 83, 84, etc.), et la classe 2 contient celles au-dessus (comme 101, 103, 105, etc.).

Étape 4 : Calculer la variance inter-classe

Pour savoir si un seuil t est bon, Otsu utilise la **variance inter-classe**, notée $\sigma_B^2(t)$. Cette grandeur mesure à quel point les deux classes sont bien séparées. Plus les moyennes $\mu_1(t)$ et $\mu_2(t)$ sont éloignées, et plus les classes ont une taille raisonnable (ni trop petites ni trop grandes), plus la variance est grande. Elle est définie par :

$$\sigma_B^2(t) = \omega_1(t) \cdot \omega_2(t) \cdot [\mu_1(t) - \mu_2(t)]^2$$

— $\omega_1(t) \cdot \omega_2(t)$: ce terme équilibre les tailles des deux classes.

— $[\mu_1(t) - \mu_2(t)]^2$: cet écart au carré mesure la différence entre les moyennes des classes.

Un seuil qui maximise cette variance donne une séparation optimale, car les classes sont alors très distinctes.

Étape 5 : Trouver le seuil optimal

On calcule $\sigma_B^2(t)$ pour chaque valeur de t entre 0 et 255, et on choisit le seuil T qui donne la plus grande variance :

$$T = \arg \max_{0 \leq t \leq 255} \sigma_B^2(t)$$

Dans notre exemple, il faudrait faire ces calculs numériquement avec les valeurs du vecteur. En pratique, T sera une intensité qui sépare bien les pixels sombres des pixels clairs, par exemple autour de 150 ou 160, mais cela dépend des résultats exacts.

Étape 6 : Binariser l'image

Une fois T déterminé, on applique ce seuil à l'image initiale pour obtenir une image binaire. Pour chaque pixel $I(x, y)$ de l'image :

— Si $I(x, y) \leq T$, on le remplace par 0 (classe sombre).

— Sinon, on le remplace par 1 (classe claire).

La règle est donc :

$$\text{Image binaire}(x, y) = \begin{cases} 0 & \text{si } I(x, y) \leq T \\ 1 & \text{sinon} \end{cases}$$

Par exemple, si $T = 150$, alors 61 devient 0, 83 devient 0, mais 203 devient 1, 206 devient 1, etc. L'image finale ne contient plus que des 0 et des 1, représentant une image en noir et blanc.

5.1.2 Pseudo-algorithme

```
// Entree : Une image en niveaux de gris I
// Sortie : Une image binaire I1

Debut
    // Etape 1 : Calcul de l histogramme et des probabilites
    Calculer l histogramme H de l image I
    Calculer la probabilit  de chaque niveau de gris P
       partir de H

    // Etape 2 : Initialisation des variables
    maxVariance <- 0
    seuilOptimal <- 0

    // Etape 3 : Parcourir tous les seuils possibles
```

```
    Pour T de 0    255 Faire
        // Calcul des probabilit s des classes
        P1 <- Somme des probabilit s de 0    T
        P2 <- 1 - P1

        // Calcul des moyennes des classes
         1  <- Moyenne des niveaux de gris de 0    T
         2  <- Moyenne des niveaux de gris de T+1    255

        // Calcul de la variance inter-classes
        variance <- P1 * P2 * ( 1  -  2 )^2

        // Mise    jour du seuil optimal si la variance est
           maximale
        Si variance > maxVariance Alors
            maxVariance <- variance
            seuilOptimal <- T
        FinSi
    FinPour

    // Etape 4 : Appliquer le seuillage optimal
    Pour chaque pixel (x, y) de l image I Faire
        Si I(x, y) >= seuilOptimal Alors
            I1(x, y) <- 255
        Sinon
            I1(x, y) <- 0
        FinSi
    FinPour

    Retourner l image segmentee I1
Fin
```

5.1.3 Résultats obtenus

Pour tester la méthode d'Otsu, nous avons choisi une image en couleur où l'objet à segmenter et l'arrière-plan présentent tous deux **des teintes distinctes**. Après avoir appliqué la procédure décrite ci-dessus, nous avons obtenu les résultats suivants :

FIGURE 5.1 – Image de départ Otsu

FIGURE 5.2 – Image segmentée Otsu

Dans cette segmentation, la méthode d'Otsu semble **relativement efficace** pour séparer le ciel de la scène principale (le sphinx et la pyramide).

Maintenant, pour une image plus complexe, contenant **plusieurs objets** et un fond aux teintes distinctes, nous avons eu les résultats suivants :

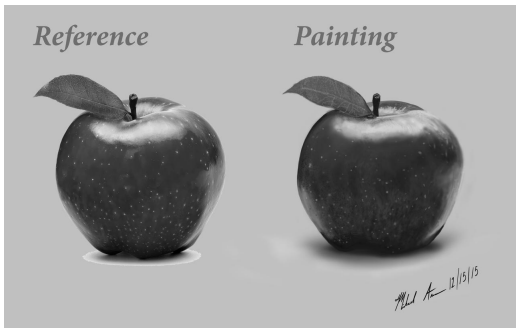

FIGURE 5.3 – Image de départ Otsu

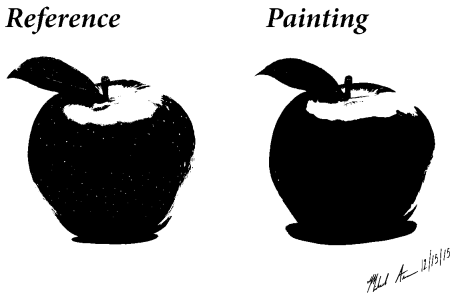

FIGURE 5.4 – Image segmentée Otsu

On peut remarquer ici que la segmentation ne parvient pas à capturer correctement **les contours et les détails fins** des pommes.

Nous avons aussi testé la méthode sur une image plus compliquée ayant des variations d'intensité et une arrière plan complexe, et voici ce que nous avons eu comme résultat :

FIGURE 5.5 – Image de départ Otsu

FIGURE 5.6 – Image segmentée Otsu

Le principal défaut de la méthode d'Otsu visible ici est la mauvaise segmentation de l'arrière-plan. Bien que le chat soit correctement extrait, des parties de l'arrière-plan ayant des intensités similaires à celles du chat ont également été conservées. Ce problème survient car Otsu utilise un seuil global, ce qui fonctionne mal lorsque l'image présente des variations d'intensité ou des arrière-plans complexes. Une approche adaptative ou multi-seuils pourrait améliorer les résultats.

5.2 Segmentation par la méthode K-means

La méthode **K-means** est une technique de classification non supervisée qui regroupe les pixels d'une image en K classes distinctes en fonction de leurs intensités. Contrairement à la méthode d'Otsu, qui repose sur l'optimisation de la variance inter-classe pour déterminer un seuil unique et binaire, K-means permet une segmentation multi-classes, offrant ainsi une meilleure adaptabilité aux images complexes

5.2.1 Explication mathématique

L'algorithme K-means vise à partitionner un ensemble de données en K clusters en minimisant la variance intra-classe définie comme :

$$J = \sum_{i=1}^{K} \sum_{x_j \in C_i} ||x_j - \mu_i||^2 \tag{5.1}$$

où :
— K est le nombre de clusters,
— C_i est le cluster i,
— x_j est un pixel appartenant au cluster C_i,
— μ_i est le centroïde du cluster C_i.

L'algorithme suit les étapes suivantes :

1. Initialiser K centroïdes (aléatoirement ou avec K-means++)
2. Affecter chaque pixel au centroïde le plus proche
3. Recalculer les centroïdes en prenant la moyenne des pixels de chaque cluster
4. Répéter les étapes 2 et 3 jusqu'à convergence (i.e., lorsque les centroïdes ne changent plus significativement).

5.2.2 Pseudo-algorithme

```
// Entree : Une image en niveaux de gris I
// Sortie : Une image segmentee I1

Debut
    // Etape 1 : Chargement et pretraitement de l'image
    Charger l'image en niveaux de gris depuis le fichier d'
       entree
```

```
    Appliquer un flou gaussien pour reduire le bruit et
       lisser les variations de l'image

    // Etape 2 : Conversion de l'image pour le traitement K-
       means
    Convertir l'image en une matrice unidimensionnelle de
       points (N, 1)

    // Etape 3 : Initialisation des centroides
    Initialiser K centroides (aleatoirement ou avec K-means
       ++)

    // Etape 4 : Execution de l'algorithme K-means
    Repeter jusqu'a convergence
        // Attribution des pixels aux centroides
        Assigner chaque pixel au centroide le plus proche en
           fonction de la distance euclidienne

        // Mise a jour des centroides
        Recalculer les centroides comme la moyenne des pixels
            appartenant a chaque cluster

        // Condition d'arret
        Verifier si les centroides ont change de maniere
           significative ; si non, arreter
    FinRepeter

    // Etape 5 : Reconstruction de l'image segmentee
        Pour chaque pixel (x, y) de l'image I Faire
            Remplacer la valeur du pixel par la valeur de son
            centroide
        FinPour

    // Etape 6 : Affichage et sauvegarde du resultat
    Afficher l'image segmentee I1 a l'ecran
    Enregistrer le resultat dans un fichier de sortie

    Retourner l'image segmentee I1
Fin
```

5.2.3 Résultats obtenus

L'algorithme K-means permet de regrouper les pixels en plusieurs classes de niveaux de gris. Voici les résultats obtenus :

FIGURE 5.7 – Image de départ K-means

FIGURE 5.8 – Image segmentée K-means

Maintenant, pour une image plus complexe, contenant **plusieurs objets** et un fond aux teintes distinctes, nous avons eu les résultats suivants :

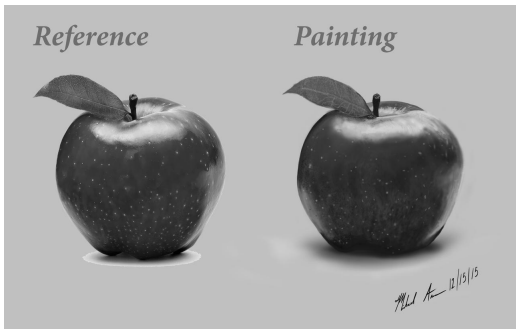

FIGURE 5.9 – Image de départ K-means

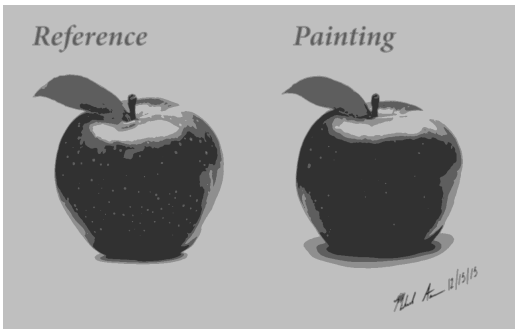

FIGURE 5.10 – Image segmentée K-means

Nous avons aussi testé la méthode sur une image plus compliquée ayant des variations d'intensité et une arrière plan complexe, et voici ce que nous avons eu comme résultat :

FIGURE 5.11 – Image de départ K-means

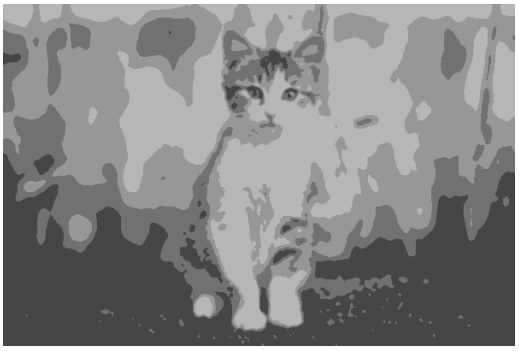

FIGURE 5.12 – Image segmentée K-means

La segmentation par K-means appliquée sur l'image en niveaux de gris permet une classification des pixels en plusieurs groupes d'intensité, ce qui offre une sépara-

tion plus détaillée des différentes zones de l'image. Cependant, elle peut générer des frontières floues et une sensibilité au choix du nombre de clusters. En comparaison, la méthode d'Otsu effectue un seuillage binaire qui segmente l'image en deux classes distinctes (objet et fond), garantissant une séparation nette mais pouvant entraîner une perte d'informations si l'histogramme des intensités n'est pas clairement bimodal. Ainsi, K-means permet une segmentation plus progressive, tandis qu'Otsu est plus direct mais parfois trop simpliste pour des images complexes.

5.2.4 Conclusion

L'algorithme K-means constitue une alternative puissante pour la segmentation d'images. Il est cependant plus complexe et nécessite une bonne initialisation des centroïdes pour obtenir des résultats optimaux. Une amélioration possible serait d'adapter K-means avec des techniques de pré-traitement avancées ou d'intégrer des méthodes d'apprentissage automatique pour ajuster dynamiquement le nombre de classes nécessaires.

Chapitre 6

Organisation du travail

6.1 Répartition des tâches

Le projet a été structuré et divisé en plusieurs phases distinctes afin d'assurer une progression méthodique et efficace :

1. **Recherche bibliographique sur les algorithmes de segmentation** : Cette phase a consisté à explorer les différentes techniques de segmentation d'images, telles que le seuillage (Otsu), les approches basées sur les contours (comme Chan-Vese), et les méthodes basées sur les régions ou les réseaux de neurones, en s'appuyant sur des articles scientifiques et des ressources techniques.
2. **Conception et choix des outils** : Nous avons évalué et sélectionné les outils les plus adaptés, notamment des bibliothèques comme OpenCV, en tenant compte de leurs fonctionnalités, compatibilités et performances pour notre implémentation.
3. **Implémentation des algorithmes** : Cette étape a impliqué le codage des algorithmes choisis, en s'assurant de leur intégration fluide dans notre environnement de développement, avec des tests préliminaires pour vérifier leur fonctionnement.
4. **Validation et optimisation des performances** : Nous avons évalué les résultats des algorithmes sur différents jeux de données, mesuré leurs performances (précision, temps d'exécution) et optimisé les paramètres pour améliorer la qualité des segmentations.
5. **Rédaction du rapport** : Enfin, nous avons synthétisé nos travaux, analysé les résultats obtenus, et documenté les choix techniques, les défis rencontrés et les perspectives d'amélioration dans un rapport final.

6.2 Difficultés rencontrées

La principale difficulté rencontrée lors de ce projet a été **l'installation et la configuration d'OpenCV en C++**, qui a exigé une gestion minutieuse des dépendances, des compilateurs et des bibliothèques spécifiques, ainsi qu'une adaptation

aux particularités de différents environnements de développement (systèmes d'exploitation, versions de logiciels). Cette tâche a été chronophage en raison des erreurs de compilation fréquentes et des incompatibilités potentielles. Par ailleurs, au début de notre travail, nous avons expérimenté l'algorithme de Chan-Vese pour la segmentation, mais les résultats obtenus étaient insatisfaisants, notamment en termes de précision et de stabilité sur nos images d'entrée, en raison de sa sensibilité aux paramètres initiaux et aux variations d'éclairage. Cela nous a conduits à explorer d'autres algorithmes, comme la méthode d'Otsu, afin d'obtenir des résultats plus robustes et adaptés à notre problème spécifique.

Conclusion

Ce projet a permis de développer et d'évaluer des algorithmes de segmentation d'images, en se concentrant initialement sur la méthode de Chan-Vese, avant d'explorer d'autres approches comme la méthode d'Otsu pour **surmonter les limites rencontrées, notamment en termes de précision et de robustesse** face aux variations d'éclairage ou au bruit.

Les étapes de conception, implémentation et validation ont démontré l'importance d'un prétraitement rigoureux des images (conversion en niveaux de gris, réduction du bruit, amélioration du contraste) pour optimiser les performances des algorithmes.

Bien que la méthode de Chan-Vese ait présenté des défis liés à sa sensibilité aux paramètres initiaux, la méthode d'Otsu a offert des résultats plus stables et rapides, bien qu'elle ait ses propres limitations, telles que la perte de détails fins. Les tests réalisés ont validé la fiabilité des classes fondamentales et des processus mis en œuvre, tout en soulignant la nécessité d'adapter les algorithmes aux spécificités des images d'entrée.

En perspective, des améliorations pourraient inclure l'intégration de techniques d'apprentissage automatique, comme les réseaux de neurones convolutifs, pour une segmentation plus précise et adaptable, ainsi qu'une optimisation supplémentaire des performances pour des applications en temps réel. Par ailleurs, notre travail parallèle sur **la détection de tumeurs cérébrales en 3D à partir d'IRM, réalisé en Python**, a permis d'obtenir des résultats prometteurs, démontrant le potentiel de ces approches pour des applications médicales avancées.

Perspectives futures

Dans le cadre de ce projet, nous avons exploré différentes approches de segmentation d'images, notamment le seuillage d'Otsu. Toutefois, pour des applications plus complexes, notamment en **imagerie médicale**, les méthodes basées sur l'apprentissage profond se révèlent beaucoup plus efficaces.

Segmentation des tumeurs cérébrales

Dans un travail en parallèle, nous avons appliqué un modèle **U-Net** pour la **segmentation des tumeurs cérébrales** sur des images IRM en utilisant la base de données **BRATS2020 (Brain Tumor Segmentation Challenge 2020)** [1]. Cette base de données comprend des images multimodales (*T1, T1ce, T2, FLAIR*) et des annotations précises des tumeurs, permettant un entraînement supervisé efficace.

Méthodologie

Les étapes principales suivies pour entraîner et évaluer le modèle sont les suivantes :

- **Prétraitement des données** :
  - Normalisation des intensités voxel par voxel.
  - Redimensionnement des images pour assurer une cohérence des dimensions d'entrée du réseau.
  - Augmentation de données (*flips, rotations, elastic transformations*) pour améliorer la robustesse du modèle.
- **Architecture du modèle** : Utilisation de **U-Net**, un réseau basé sur une structure encodeur-décodeur avec connexions résiduelles, permettant une segmentation pixel-par-pixel.
- **Entraînement et validation** :
  - Fonction de perte : combinaison de **Dice Loss** et **Binary Cross Entropy (BCE)** pour optimiser la segmentation.
  - Utilisation d'un **scheduler de learning rate** et de l'optimisation par **Adam**.
  - Évaluation avec les métriques **Dice Score** et **Hausdorff Distance**.

Résultats obtenus

Le modèle U-Net a permis une segmentation efficace des trois sous-régions tumorales :

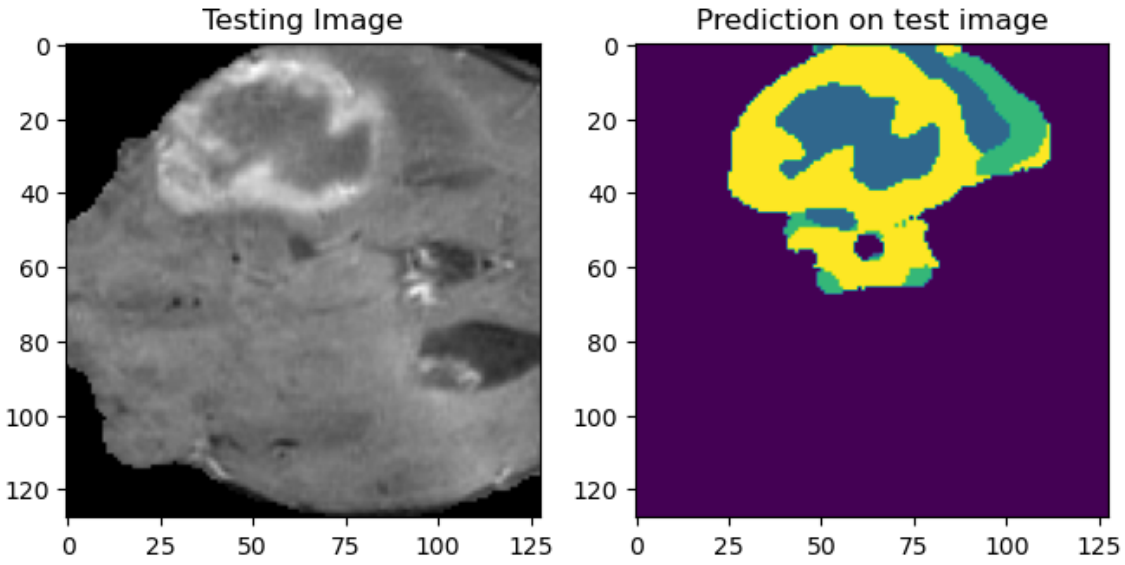

FIGURE 6.1 – Résultat segmentation UNet

Lien avec le projet en C++

Dans ce projet, nous avons utilisé des approches plus classiques comme Otsu. Toutefois, ces techniques montrent **leurs limites** lorsqu'il s'agit de segmenter des structures complexes comme des tumeurs cérébrales, où les variations d'intensité et de texture rendent difficile un simple seuillage global.

Une **évolution naturelle** de ce projet serait d'**intégrer U-Net dans un environnement C++** pour permettre une segmentation avancée directement depuis une application en temps réel. Cela pourrait être réalisé via :

- L'intégration de **TensorFlow C++ API** ou **ONNX Runtime** pour exécuter le modèle entraîné.
- L'utilisation de **CUDA** pour optimiser l'inférence sur GPU et permettre un traitement rapide des images médicales.
- Une interface graphique (Qt ou OpenCV GUI) pour visualiser les résultats en temps réel.

Ces améliorations ouvriraient la voie à une application clinique utilisable dans des **systèmes d'aide au diagnostic** pour les radiologues et neurologues.

Bibliographie

[1] Awsaf. "BraTS2020 Dataset (Training + Validation)". In : *Brain Tumor Segmentation 2020 Dataset, kaggle* (). URL : https://www.kaggle.com/datasets/awsaf49/brats20-dataset-training-validation.

[2] Nicolas Loménie. "Image Processing and Analysis - Introduction for Computational Biology". In : *Paris Descartes* (). URL : https://helios2.mi.parisdescartes.fr/~lomn/Cours/BC/Publis/CompBio4.pdf.