Analyse et Segmentation d'Images Microscopiques

Réalisé par : Mathis Aulagnier
Cours : VISION ARTIFICIELLE ET TRAITEMENT DES IMAGES (8INF804)

1 Introduction

L'objectif de ce projet est de segmenter et d'isoler chaque grain dans une image, afin de faciliter l'analyse de leurs caractéristiques individuelles. En extrayant ces caractéristiques, et avec des labels d'identification pour chaque minéral détecté, il serait ensuite possible d'utiliser des algorithmes de classification, comme les arbres de décision ou d'autres méthodes d'apprentissage automatique. Cela permettrait de développer un système de détection automatique des minéraux, capable de distinguer divers types de grains de façon précise et autonome.

2 Méthodologie

Pour présenter la méthodologie, je vais retracer ici les étapes suivies dans le traitement de l'image.

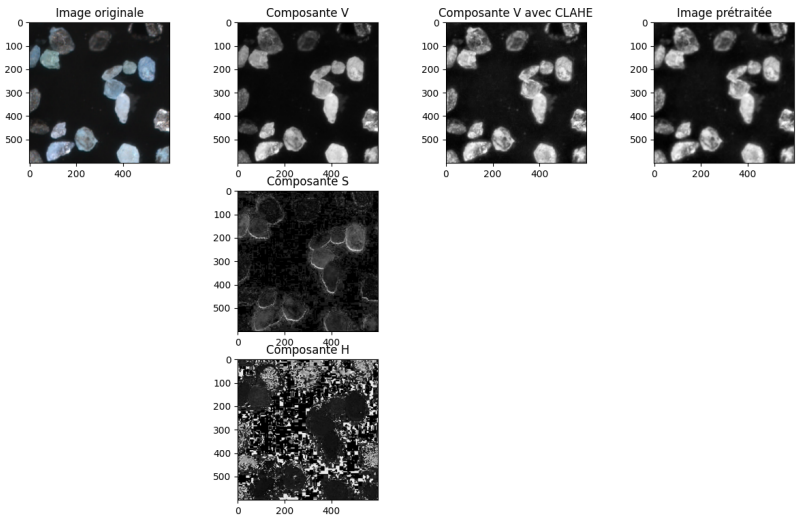

Figure 1: Étapes du prétraitement

L'image d'origine est visible sur la première image à gauche. J'ai d'abord converti cette image dans l'espace de couleur HSV, ce qui m'a permis d'extraire la composante *Value* (V). Cette composante de luminance, m'a semblé plus pertinente que celles de *Hue* (teinte) et de *Saturation* pour l'analyse des grains. J'ai également essayé une approche dans l'espace de couleur LAB, mais les résultats obtenus étaient légèrement moins satisfaisants. J'ai également envisagé d'utiliser simplement l'image en niveaux de gris, qui donnait de bons résultats. Cependant, j'ai décidé de garder la composante V de l'espace HSV, car elle est souvent plus efficace pour la détection d'objets.

L'analyse des image a révélé que certains grains étaient plus sombres par rapport à d'autres qui étaient plus lumineux. J'ai donc amélioré le contraste en appliquant l'algorithme de Contrast Limited Adaptive Histogram Equalization (CLAHE). Le résultat est visible dans la troisième image de la figure 2. Cet ajustement de contraste a permis de faire ressortir les détails des grains. Enfin, pour réduire le bruit, j'ai appliqué un filtre gaussien, ce qui est visible dans la transition de la troisième à la quatrième image. Cela a permis d'atténuer le bruit, même si la différence est peu perceptible visuellement.

Le prétraitement étant terminé, j'ai pu commencer la segmentation en utilisant l'algorithme Watershed. Cette méthode repose sur la création de **bassins** autour de marqueurs définis par l'utilisateur, qui servent de points de départ pour inonder chaque région jusqu'à ce qu'elles se rencontrent, formant ainsi des lignes de séparation entre les grains. Je me suis appuyé sur la figure suivante pour illustrer les étapes.

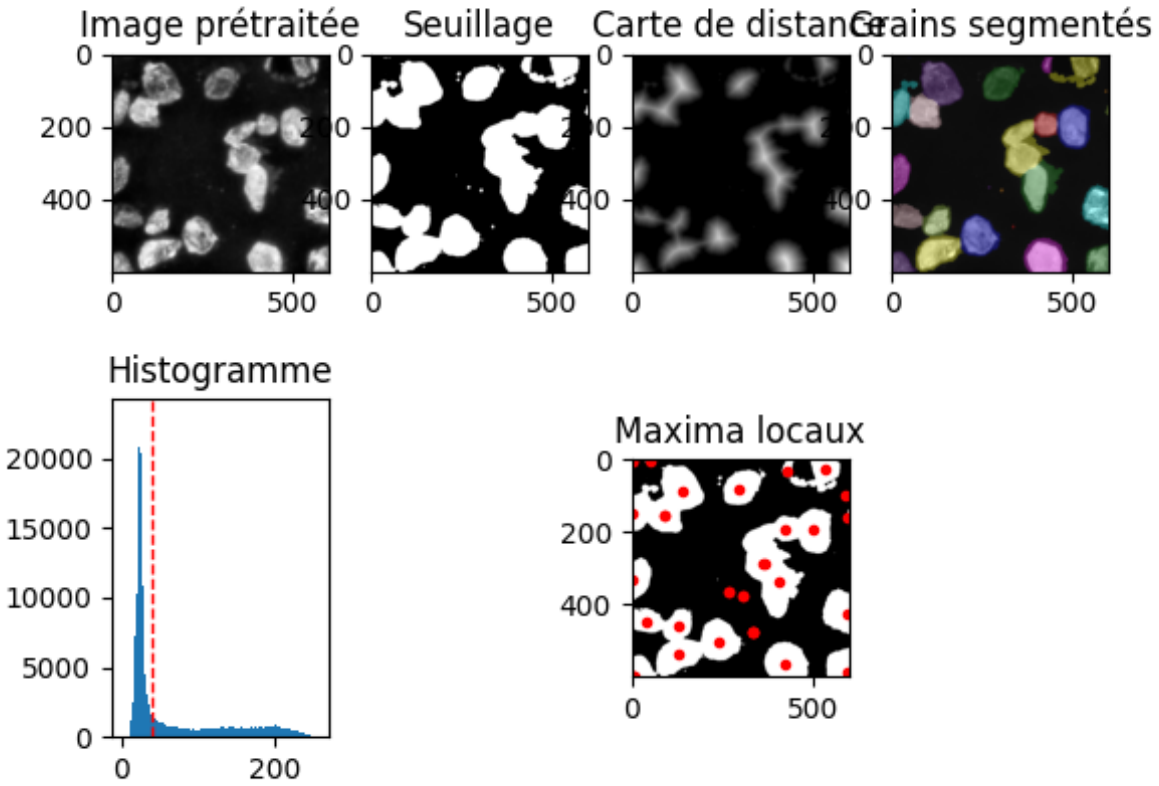

Figure 2: Étapes de la segmentation par Watershed

À partir de l'image prétraitée, j'ai cherché à binariser les grains (c'est-à-dire à les distinguer clairement les uns des autres). Cette étape m'a permis de me positionner au même point de départ que celui du tutoriel disponible dans les ressources Watershed segmentation. Pour cela, j'ai d'abord appliqué un seuillage afin de créer une image binaire. Après plusieurs essais, les méthodes de seuillage adaptatif et OTSU n'ont pas donné de résultats plus satisfaisants qu'un simple seuillage à partir d'une valeur fixe. J'ai donc analysé l'histogramme de l'image et sélectionné un seuil fixe de 40 (indiqué par une ligne rouge sur l'histogramme), ce qui a permis d'obtenir un résultat concluant et d'accélérer l'exécution du code.

Une fois le seuillage appliqué, j'ai calculé une carte des distances, visible dans la troisième colonne de la figure 2. Cette carte des distances indique, pour chaque pixel, la distance par rapport au pixel de fond le plus proche. Les pixels de fond correspondent aux pixels noirs de notre image binarisée. Les pixels les plus éloignés du fond apparaissent comme des *sommets* sur cette carte, représentés en blanc. Plus un pixel est clair, plus il est éloigné du fond noir.

Après avoir identifié ces *sommets*, nous avons défini des marqueurs, également visibles dans la troisième colonne de la figure 2. Ces marqueurs correspondent aux points les plus éloignés du fond, aussi appelés *maxima locaux* ou *sommets*, et servent de points de départ pour l'algorithme Watershed. Placés au niveau des pixels les plus élevés (les maxima locaux), ils représentent ainsi les centres des futurs bassins.

L'algorithme Watershed démarre alors l'**inondation** depuis chaque marqueur, qui se propage dans toutes les directions jusqu'à ce que deux zones inondées se rencontrent. Ce processus crée ce que l'on appelle des bassins dans l'algorithme Watershed. Aux points de rencontre entre les zones inondées, une ligne de séparation se forme, marquant la frontière entre les objets. Ainsi, les lignes de séparation apparaissent naturellement là où les bassins se rejoignent, segmentant les grains de manière efficace.

Le code suivant illustre la mise en place des marqueurs et l'application de l'algorithme Watershed. J'ai utilisé une matrice de marqueurs où les coordonnées des maxima locaux servent de départ pour chaque bassin :

```
markers = np.zeros_like(thresh_g, dtype=bool)
markers[tuple(coordinates.T)] = True
markers = ndi.label(markers)[0]

# Application du Watershed
labels = watershed(-distance, markers, mask=thresh_g)
image_colorier = color.label2rgb(labels, image=gray_image, bg_label=0)
```

3 Difficultés rencontrées

Durant ce projet, j'ai été confronté à plusieurs difficultés techniques, notamment liées à la gestion de la luminosité, du contraste et du bruit résiduel dans les images.

Une autre difficulté a été de régler les paramètres des différentes opérations de traitement d'image pour obtenir une détection optimale.

La compréhension et la mise en œuvre de l'algorithme Watershed ont également constitué un défi. Cet algorithme exige une bonne maîtrise des concepts de marqueurs et de distance, ainsi que de l'influence de chaque paramètre sur le processus d'inondation et la formation des lignes de séparation.

Enfin, pour améliorer mon code, j'ai tenté d'intégrer l'algorithme SLIC. Avant l'application de Watershed, j'ai cherché à segmenter l'image en superpixels avec SLIC afin d'obtenir des frontières plus nettes autour des grains. Les superpixels générés par SLIC créent une segmentation initiale plus homogène, ce qui facilite ensuite la séparation par Watershed. Cependant, même en augmentant fortement le paramètre `n_segments` (nombre de superpixels), l'algorithme SLIC n'a pas pu détecter tous les grains et ne segmentait qu'un seul. Malgré mes tentatives d'ajustement de sensibilité, les résultats sont restés insuffisants pour obtenir une détection complète. J'ai donc finalement décidé de me concentrer uniquement sur l'algorithme Watershed pour ce projet.

4 Résultats et conclusion

Les performances de l'algorithme de segmentation des grains se sont révélées satisfaisantes. Grâce aux étapes de prétraitement, incluant l'utilisation de la composante V dans l'espace de couleur HSV et la normalisation par CLAHE, l'algorithme a pu réduire les effets des variations de luminosité.

En termes de résultats quantitatifs, l'algorithme a détecté correctement 321 grains sur les 372 grains réels, atteignant ainsi une précision de 86%. Le tableau suivant détaille les résultats par image :

Image	Grains détectés	Fausses détections	Grains réels	Accuracy
fig 1	34	5	39	87.18%
fig 2	29	13	53	54.72%
fig 3	44	5	49	89.80%
fig 4	39	6	45	86.67%
fig 5	41	3	44	93.18%
fig 6	37	2	39	94.87%
fig 7	26	1	27	96.30%
fig 8	30	4	34	88.24%
fig 9	31	11	42	73.81%

Table 1: Résultats de détection des grains par image

Malgré quelques erreurs de détection, les erreurs sont principalement dues à une détection d'un seul grain alors qu'il y en a en réalité plusieurs, l'algorithme a montré une bonne capacité à isoler les grains individuels dans les images variées.

En conclusion, ce projet m'a permis de développer une approche efficace pour la segmentation de grains dans des images, malgré certaines limitations dues à des contrastes ou expositions difficiles. Dans le futur, l'algorithme pourrait être amélioré pour augmenter sa précision et mieux gérer les grains trop proches les uns des autres.

Annexe : Code Python

Voici le code Python utilisé dans le cadre de ce projet :

```
import cv2
import numpy as np
import pandas as pd
from skimage import color
from skimage.segmentation import watershed
from skimage.feature import peak_local_max
from scipy import ndimage as ndi
import matplotlib.pyplot as plt

def preprocess_image(image):
    """Pr traitement de l'image."""
    # Conversion en niveaux de HSV
    HSV = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    # Extraire la composante de luminance
    h, s, v = cv2.split(HSV)
    value = v
    # Am lioration du contraste
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
    value_clahe = clahe.apply(value)
    # R duction du bruit
    preprocessed_image = cv2.GaussianBlur(value_clahe, (9, 9), 0)
    return preprocessed_image

def watershed_segmentation(preprocessed_image, gray_image):
    """Segmentation des grains par Watershed."""
    # Seuillage
    _, thresh_g = cv2.threshold(preprocessed_image, 40, 255, cv2.THRESH_BINARY)

    # Cr er une carte de distance
    distance = ndi.distance_transform_edt(thresh_g)
    coordinates = peak_local_max(distance, footprint=np.ones((65, 65)), labels=thresh_g)

    # Marquer les grains
    markers = np.zeros_like(thresh_g, dtype=bool)
    markers[tuple(coordinates.T)] = True
    markers = ndi.label(markers)[0]

    # Application du watershed
    labels = watershed(-distance, markers, mask=thresh_g)
    image_colorier = color.label2rgb(labels, image=gray_image, bg_label=0)

    return labels, image_colorier

def extract_grain_features(image, gray_image, labels):
    # Calcul des moyennes BGR pour chaque segment de grain
    data = {"Moyenne de B": [], "Moyenne de G": [], "Moyenne de R": []}

    # Boucle sur chaque grain d tect
    for label in np.unique(labels):
        if label == 0:  # Ignorer le fond
            continue
        mask = np.zeros_like(gray_image, dtype="uint8")
        mask[labels == label] = 255

        # Calcul des moyennes de chaque canal BGR dans la zone du masque
        mean_val = cv2.mean(image, mask=mask)
        data["Moyenne de B"].append(mean_val[0])
        data["Moyenne de G"].append(mean_val[1])
        data["Moyenne de R"].append(mean_val[2])

    # Cr ation du DataFrame
    df = pd.DataFrame(data)
    df.index = [f"Grain{i+1}" for i in range(len(df))]
    return df

def main(img_path, show_images=False):
    # Charger l'image
    image_path = img_path  # Remplace par ton chemin d'image
```

```
    image = cv2.imread(image_path)

    if image is None:
        raise ValueError("Impossible de charger l'image. V rifiez le chemin.")

    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    preprocessed_image = preprocess_image(image)

    # Segmentation Watershed
    labels, image_colorier = watershed_segmentation(preprocessed_image, gray_image)

    # Extraction des caract ristiques
    features_df = extract_grain_features(image, gray_image, labels)

    print(features_df)

    # Affichage des r sultats
    if show_images:
        plt.subplot(1, 3, 1)
        plt.imshow(image)
        plt.title("Image originale")
        plt.subplot(1, 3, 2)
        plt.imshow(preprocessed_image, cmap="gray")
        plt.title("Image pr trait e")
        plt.subplot(1, 3, 3)
        plt.imshow(image_colorier)
        plt.title("Grains segment s")
        plt.show()

# dir_path = "Images/"
# images_names = [
#     "Echantillion1Mod2_301.png",
#     "Echantillion1Mod2_302.png",
#     "Echantillion1Mod2_303.png",
#     "Echantillion1Mod2_304.png",
#     "Echantillion1Mod2_305.png",
#     "Echantillion1Mod2_306.png",
#     "Echantillion1Mod2_316.png",
#     "Echantillion1Mod2_422.png",
#     "Echantillion1Mod2_471.png"
# ]

if __name__ == "__main__":
    # for img_name in images_names:
    #     main(dir_path + img_name, show_images=True)
    main("Images/Echantillion1Mod2_316.png", show_images=True)
```

Listing 1: Code python