ÉCOLE NATIONALE SUPÉRIEURE DE TECHNIQUES AVANCÉES

[CSC_4MI06_TA] - [COMPLÉMENTS POUR L'ANALYSE D'IMAGES]

Rapport sur le TP2 - Détection des veines oculaires

Professeurs:
Antoine MANZANERA

Étudiants :
Gabriel BAPTISTA TRELLESSE
Rodrigo BOTELHO ZUIANI

30 de maio de 2025

Sommaire

1 Introduction

Le but du projet est de proposer une manière de segmenter des réseaux vasculaires dans les images de rétine SLO (Scanning Laser Ophthalmoscopy). Les figures 1 et 2 montrent un exemple.

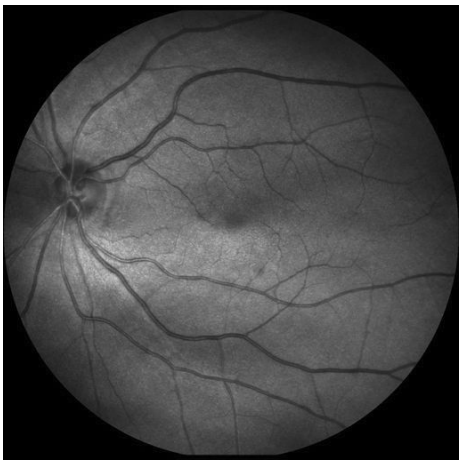

Figura 1: Image SLO

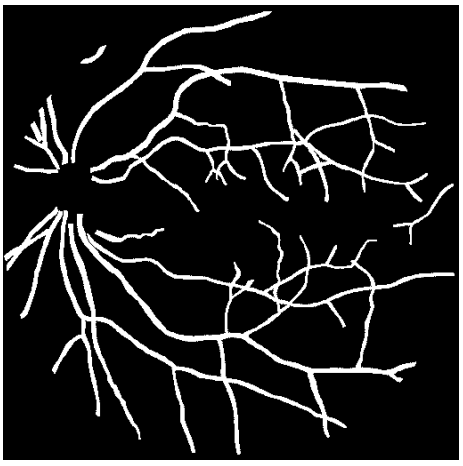

Figura 2: Marcation qui sert comme vérité terrain

Dans ce document, on va présenter 3 méthodes utilisés pour atteindre l'objectif, une explication de chacun et une comparaison des résultats.

2 Méthodologie

2.1 Algorithme WaterShed

Cette méthode utilise un rehaussement de contraste, opérations morphologiques, détection de contours et enfin une segmentation par lignes de partage des eaux, le watershed.

Tout d'abord, on normalise l'image, en mettant les valeurs de tous les pixels entre 0 et 1. Puis on utilise la fonction 'equalize_adapthist', qui rehausse le contraste dans les zones sombres et claires, pour mieux distinguer les vaisseaux. Elle divisant l'image en parties égales, et utilise une CLAHE (Contrast Limited Adaptive Histogram Equalization) pour ajuster l'intensité des pixels conformément à la distribution locale des valeurs des pixels. Après, elle rejoint les parties divisées de l'image en utilisant une interpolation 'bilinea' pour éliminer les divisions entre les parties.

```
    img /= 255.0  # Normalize to [0, 1]
    img = exposure.equalize_adapthist(img, clip_limit=3)  # CLAHE
```

Ensuite, on utilise une réduction de bruit avec un flou gaussien et un filtre médian. Le flou gaussien supprime le bruit de haute fréquence, en réglant la valeur de chaque pixel de l'image sur la moyenne de valeurs de tous le pixels présents dans le rayon. Le filtre médian (non linéaire) supprime le bruit impulsionnel, en remplaçant la valeur de chaque pixel par la valeur médiane de son voisinage.

```
    img = filters.gaussian(img, sigma=0.40)
    img = filters.rank.median(img, disk(3))
```

Pour la troisième étape, on réalise une transformation top-hat noire. Elle réalise une différence entre la fermeture de l'image e la propre image, ce qui extrait des éléments petits dans l'image (plus petits que l'élément structurant) et met en évidence les éléments sombres (vaisseaux) sur le fond clair. Ici, on la réalise 2 fois avec des éléments structurants différents, un rectangle et une ellipse, pour capturer des orientations variées des vaisseaux.

```
    black_hat_img = black_tophat(img, ellipse(2,1))  # Apply black
   tophat with disk kernel
    black_hat_img2 = black_tophat(img, footprint_rectangle((1,10)))
```

Puis, on applique une détection des contours, en utilisant le filtre 'sobel'. Ce filtre extrait les contours en calculant la variation d'intensité des pixels, c'est à dire, le gradient. Pour faire ça, il utilise 2 matrices de convolution 3x3, pour calculer des approximations des dérivées horizontale et verticale. De cette façon, il obtient les

bords verticals et horizontals de l'image. Ici, on cherche à capturer les contours des vaisseaux qui produisent des gradients élevés.

```
    gradient = filters.sobel(black_hat_img)
    gradient2 = filters.sobel(black_hat_img2)
```

Ensuite, on applique une seuillage pour obtenir une représentation binaire des vaisseaux. Pour choisir la valeur de seuillage, on utilise la fonction 'threshold_otsu', qui trouve un seuil optimal. Il s'agit d'un algorithme qui cherche exhaustivement le seuil qui minimise la variance intra-classes. On combine finalement les cartes binaires issues de chaque filtre top-hat réalisés antérieurement. On élimine aussi des petits éléments, qui sont des faux positifs avec la fonction 'remove_small_objects'. On réalise aussi une remplissage de trous avec la fonction 'binary_fill_holes', pour rendre les vaisseaux continus.

```
    binary = gradient > 0.95*filters.threshold_otsu(gradient)
    binary2 = gradient2 > filters.threshold_otsu(gradient2)
    # Combine the two binary images
    binary = (binary | binary2)
    binary = remove_small_objects(binary, min_size=32)
    binary = ndi.binary_fill_holes(binary)
```

Finalement, on commence a préparer l'image pour watershed. Premièrement, on applique une transformé de distance, un filtre qui applique dans chaque pixel 1 (vaisseaux) sa distance au plus proche pixel 0 (fond). Comme ça, les centres des vaisseaux auront des valeurs plus grands. Avec ça, on applique la fonction 'local_maxima', qui trouve les maximums locaux, ce qui est équivalant aux points centraux des vaisseaux, et on donne des étiquettes différentes à chaque région connectée avec la fonction 'label'. Enfin, on applique la segmentation par lignes de partage des eaux. Le watershed "inonde"les zones à partir des maxima locaux jusqu'à rencontrer d'autres régions, séparant les structures connectées.

```
    distance = ndi.distance_transform_edt(binary)
    local_maxi = local_maxima(distance)
    markers = ndi.label(local_maxi)[0]
    labels = segmentation.watershed(-distance, markers, mask=binary
   )
    img_out = (labels > 0)
```

2.2 Algorithme Frangi

Cet algorithme utilise quelques idées similaires au méthode précédent, mais il cible directement les structures vasculaires grâce au filtre de Frangi, qui met en évidence les structures tubulaires des images.

On commence par un rehaussement de contraste, avec la fonction 'create-CLAHE'. Elle utilise aussi le CLAHE, qui a été expliqué précédemment. Puis on normalise l'image comme avant, pour mettre les valeurs des pixels entre 0 et 1, ce qui est nécessaire pour le filtre de Frangi.

```
    clahe = cv.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
    enhanced = clahe.apply(img)
    image_norm = (enhanced - enhanced.min()) / (enhanced.max() -
enhanced.min())
```

Comme avant, on applique aussi un filtre gaussien avec la fonction 'gaussien_filter', pour réduire le bruit de l'image. Ce filtre de réduction de bruit a aussi été expliqué précédemment. Avec cette image, on applique le filtre de Frangi, qui est responsable pour détecter les structures tubulaires. Tout d'abord, ce filtre calcule la matrice Hessienne pour chaque pixel, avec l'objectif d'analyser les variations de courbure locale de l'image. Pour chaque matrice calculé, il extrait les valeurs propres λ_1 et λ_2, et ces valeurs permettent d'analyser la forme locale. Ici, on cherche les formes allongées. Le 'sigmas=(1, 3)' signifie que le Frangi va appliquer tout ce processus dans 3 échelles de flou gaussien, pour détecter des vaisseaux de différents tailles.

```
    smoothed = gaussian_filter(image_norm, sigma=1)
    vessels = frangi(smoothed, sigmas=(1, 3), scale_step=0.5)
```

Comme le filtre de Frangi donne souvent des valeurs avec un faible contraste, on étend les valeurs d'intensité de l'image avec la fonction 'rescale_intensity'. Elle allonge les valeurs des pixels pour qu'ils s'adaptent à la gamme donné (0, 1). Puis on applique une seuillage simple pour obtenir une image binaire des vaisseaux.

```
    vessels_rescaled = rescale_intensity(vessels, in_range='image',
    out_range=(0, 1))
    vessels_norm = (vessels_rescaled * 255).astype(np.uint8)
    _, binary = cv.threshold(vessels_norm, 20, 255, cv.
THRESH_BINARY)
```

Finalement, on utilise comme avant la fonction 'remove_small_objects', qui a déjà été expliquée. On cherche encore une fois a éliminer des faux positifs.

```
    binary_cleaned = remove_small_objects(binary.astype(bool),
min_size=30)
    img_out = (img_mask & binary_cleaned)
```

2.3 Mathématique morphologique et topologie

La dernière solution que nous avons essayée est basée sur (ROSSANT et al., 2011), où les techniques utilisées sont la morphologie mathématique et la

topologie.

Le processus a été divisé en 3 parties :

1. Pré-traitement
2. Extraction des attributs
3. Seuillage et fusion

Pour la première partie, on vérifie si l'image d'entrée a des dimensions 512×512, afin de pouvoir utiliser les mêmes valeurs des éléments structurants que dans l'article. Une fois que les tailles sont similaires, on applique un filtre de débruitage — dans ce cas, un filtre gaussien — suivi d'une ouverture morphologique pour que les composantes connexes des vaisseaux ne soient pas affectées. Cela nous donne une image intermédiaire g.

```
    img = filters.gaussian(img, sigma=0.5)
    g = opening(img, disk(1))
```

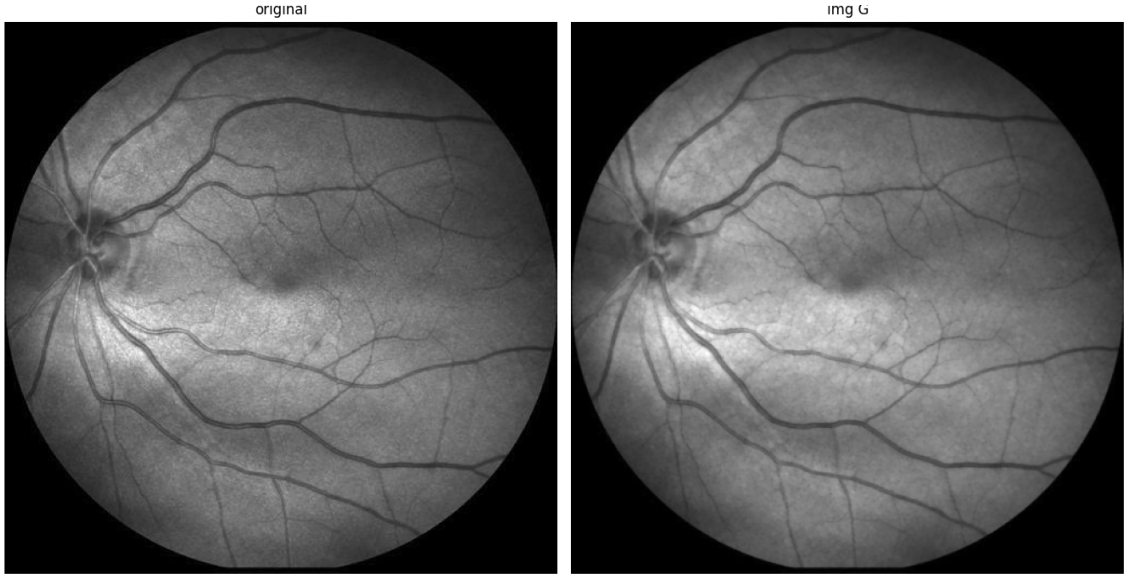

Figura 3: Images originales et image g après application du filtre gaussien et ouverture.

Après le pré-traitement, on utilise un *toggle mapping filter* pour que, dans les zones où l'ouverture a plus d'effet, elle soit appliquée pour retirer les petits bruits clairs (plus petits que l'élément structurant). Mais si la fermeture a plus d'effet, alors elle est appliquée pour remplir les éléments sombres plus petits que l'élément

structurant. Cela signifie que l'on utilise l'ouverture pour réduire les bruits blancs et la fermeture pour compléter les vaisseaux.

Le filtre est le suivant :

$$h(x) = [TM(g)](x) = \begin{cases} \phi_{B_1}(g)(x) & \text{si } \phi_{B_1}(g)(x) - g(x) \leq g(x) - \gamma_{B_2}(g)(x) \\ \gamma_{B_2}(g)(x) & \text{sinon} \end{cases}$$

```
    g_opening = opening(g, disk(3))
    g_closing = closing(g, disk(4))

    # Toggle mapping filter
    diff_open = np.abs(g - g_opening)
    diff_close = np.abs(g_closing - g)

    # Create output image
    h = np.where(diff_close <= diff_open, g_closing, g_opening)
```

Ce filtre nous donne l'image H suivante :

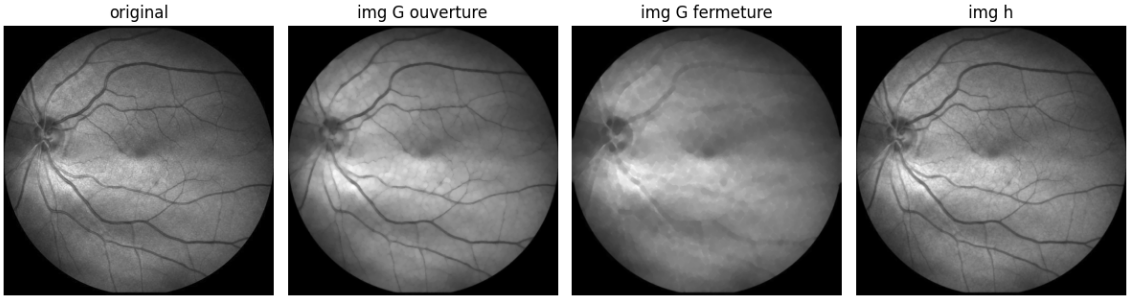

Figura 4: Image H générée par le *toggle mapping filter*.

La deuxième partie se concentre sur l'extraction des caractéristiques en utilisant le *top-hat* de H, appelé *att0*, car le top-hat permet de faire ressortir les vaisseaux tout en mettant les pixels de l'arrière-plan à zéro.

Pour extraire les caractéristiques, on utilise deux filtres `log_filter` (négatif du Laplacien du Gaussien) avec des σ différents pour extraire des caractéristiques à différentes échelles. La fonction de filtre est la suivante :

```
    def LoG_filter(image, sigma, size=None):
        # Generate LoG kernel
        if size is None:
            size = int(6 * sigma + 1) if sigma >= 1 else 7

```

```
        if size % 2 == 0:
            size += 1

        x, y = np.meshgrid(np.arange(-size//2+1, size//2+1), np.
    arange(-size//2+1, size//2+1))
        kernel = -(1/(np.pi * sigma**4)) * (1 - ((x**2 + y**2) / (2
     * sigma**2))) * np.exp(-(x**2 + y**2) / (2 * sigma**2))
        kernel = kernel
        # Perform convolution
        result = ndi.convolve(image, kernel)

        return result
```

Avec les caractéristiques extraites par les LoG, on applique l'algorithme `path_opening`, qui correspond à des ouvertures utilisant des éléments structurants en forme de ligne à différents angles, car les composantes connexes des réseaux ne sont pas uniquement horizontales ou verticales.

L'algorithme est le suivant :

```
    def path_opening(img, length=20, angle_step=15):
        """
        Improved path opening for vessel detection

        Parameters:
        - img: input binary image
        - length: maximum length of paths to consider
        - angle_step: angular resolution (degrees) for directional
    structuring elements

        Returns:
        - Result of path opening operation
        """
        # Normalize input to [0,1]
        img = img.astype(np.float32)
        if img.max() > 1:
            img = img / 255.0

        # Create structuring elements for multiple directions
        results = []
        angles = np.arange(0, 180, angle_step)

        for angle in angles:
            # Create line structuring element
            if angle == 0:
                se = footprint_rectangle((1, length))
            elif angle == 90:
                se = footprint_rectangle((length, 1))
            else:
```

```
                # For other angles, we'll create a rotated line
                theta = np.radians(angle)
                x = np.round(length/2 * np.cos(theta)).astype(int)
                y = np.round(length/2 * np.sin(theta)).astype(int)
                xx, yy = np.meshgrid(np.arange(-x, x+1), np.arange
(-y, y+1))
                mask = np.abs(yy - xx * np.tan(theta)) < 0.5
                se = mask.astype(np.uint8)
            if np.sum(se) == 0:
                continue
            # Apply directional opening (erosion followed by
dilation)
            opened = dilation(erosion(img, se), se)
            results.append(opened)

        # Combine results from all directions
        combined = np.maximum.reduce(results)

        # Post-processing to clean up small artifacts
        combined = opening(combined, disk(1))

        return combined
```

Le path_opening nous donne les caractéristiques, mais malheureusement, nous n'avons pas réussi à fusionner les deux avec un seuillage par hystérésis et une ouverture algébrique comme développé dans l'article, car nous avons rencontré des difficultés avec les formulations mathématiques pour définir les seuils haut et bas. Nous avons donc terminé en prenant simplement le maximum entre les deux cartes de caractéristiques : *att1* pour le plus petit σ et *att2* pour le plus grand.

```
    # Attribute Extraction
    att0 = invert(h) - opening(invert(h), disk(dmax/2 + 2)) #
background --> 0

    sigma1 = dmax/6
    sigma2 = dmax/4
    att1 = LoG_filter(att0, sigma1)
    att2 = LoG_filter(att0, sigma2)
    thresh1 = filters.threshold_otsu(att1)
    binary_att1 = att1 > thresh1
    thresh2 = filters.threshold_otsu(att2)
    binary_att2 = att2 > thresh2
    combined_att = np.maximum(att1, att2)
    rst1 = path_opening(combined_att, length=10, angle_step=45)
    threshrst = filters.threshold_otsu(rst1)
    binary_rst1 = rst1 > threshrst
```

Et cela nous a donné les résultats ci-dessous :

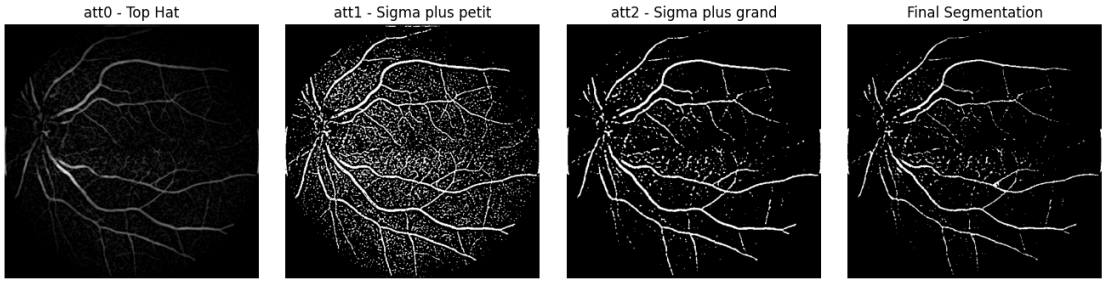

Figura 5: Caractéristiques intermédiaires et segmentation finale.

Ainsi, *att1* nous donne les caractéristiques à petite échelle et *att2* celles à grande échelle. Lorsque l'on utilise les deux, on obtient une carte du réseau avec peu de bruit et des vaisseaux bien segmentés.

3 Comparaison des résultats

On a appliqué ces 3 méthodes à chacune des images pour faire une comparaison des résultats. On peut les comparer visuellement, avec les outputs, et aussi en comparant les valeurs de *precision*, *recall*, et *F1*.

3.1 Résultats visuels pour 10 images

On vas présenter les segmentations et leurs squelette pour une colleta de 10 images.

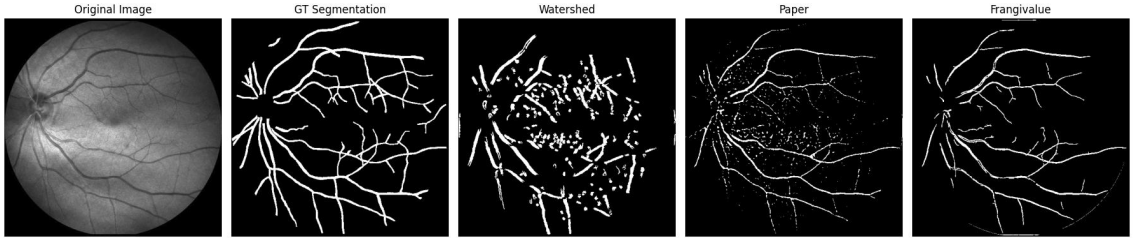

Figura 6: Segmentations pour l'image 1

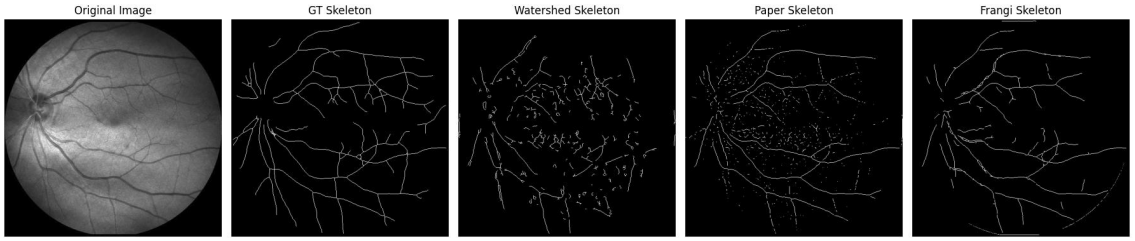

Figura 7: Squelettes pour l'image 1

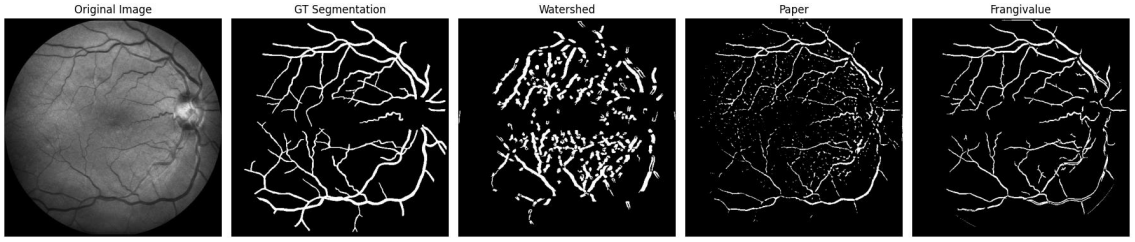

Figura 8: Segmentations pour l'image 2

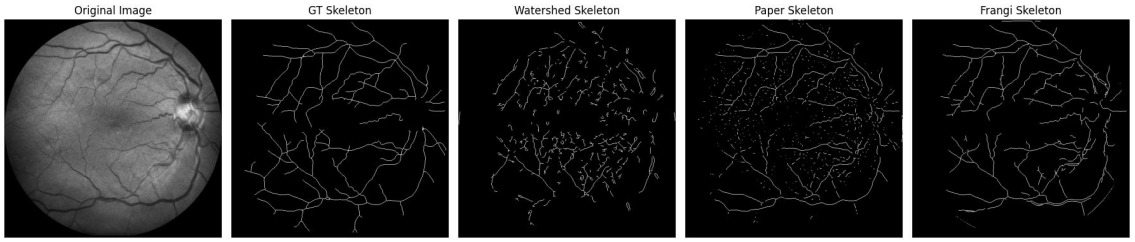

Figura 9: Squelettes pour l'image 2

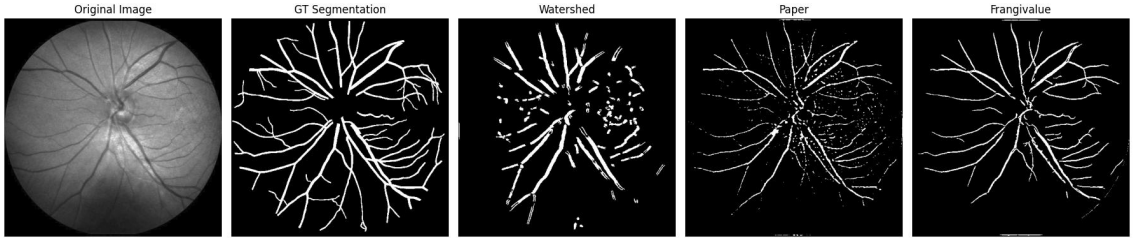

Figura 10: Segmentations pour l'image 3

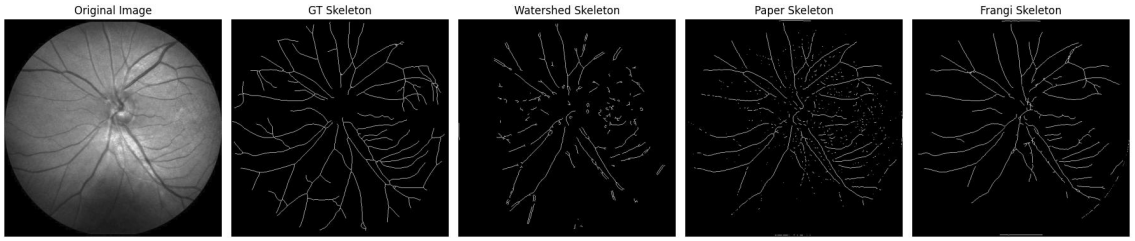

Figura 11: Squelettes pour l'image 3

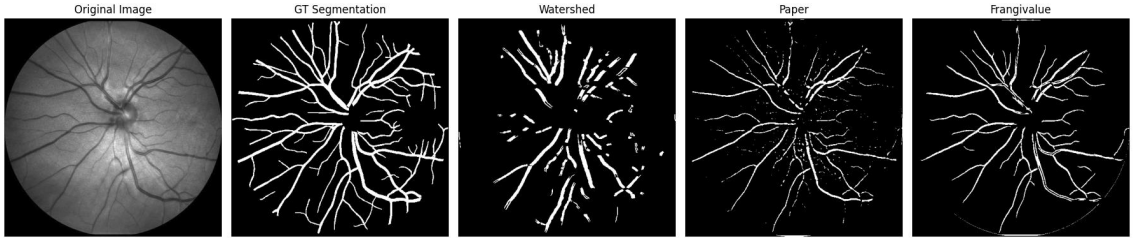

Figura 12: Segmentations pour l'image 4

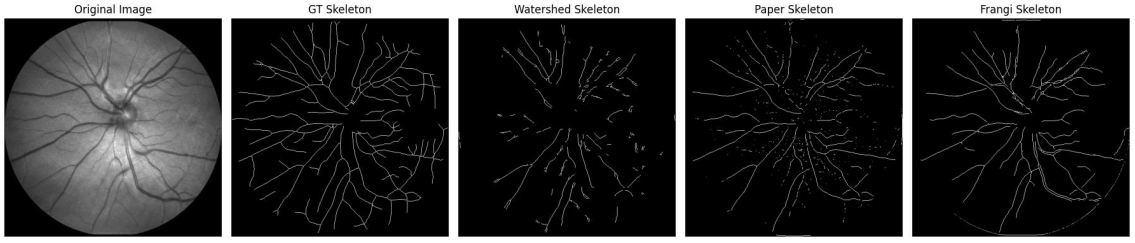

Figura 13: Squelettes pour l'image 4

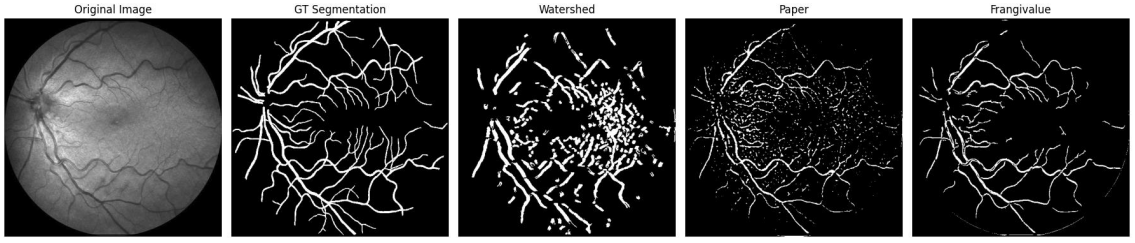

Figura 14: Segmentations pour l'image 5

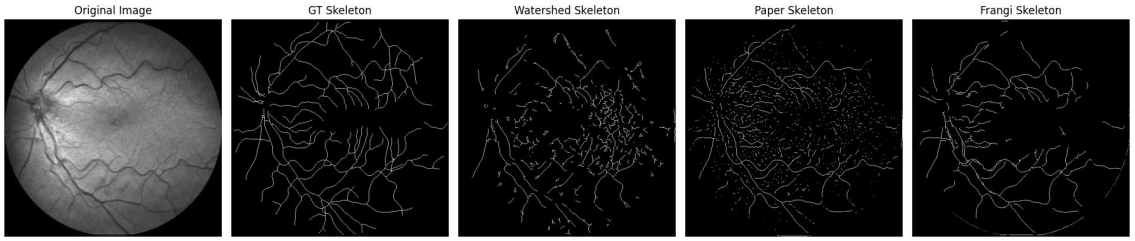

Figura 15: Squelettes pour l'image 5

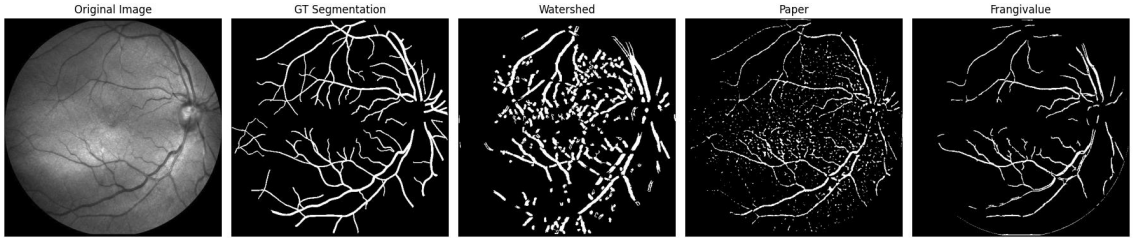

Figura 16: Segmentations pour l'image 6

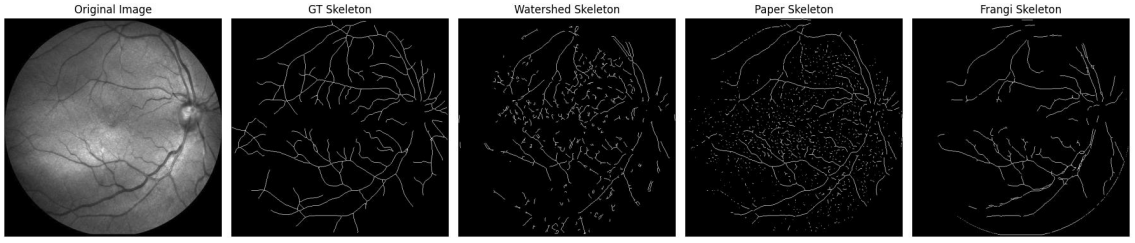

Figura 17: Squelettes pour l'image 6

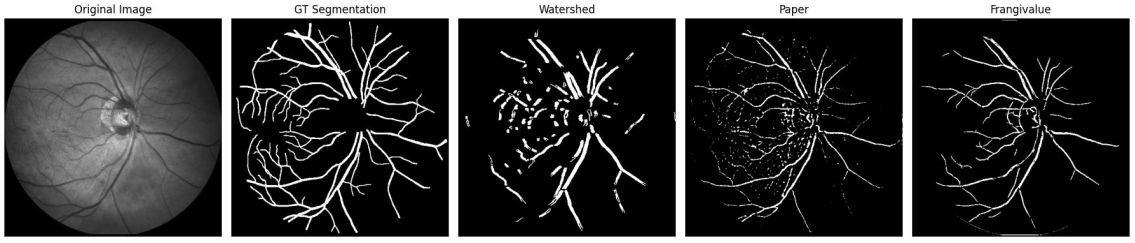

Figura 18: Segmentations pour l'image 7

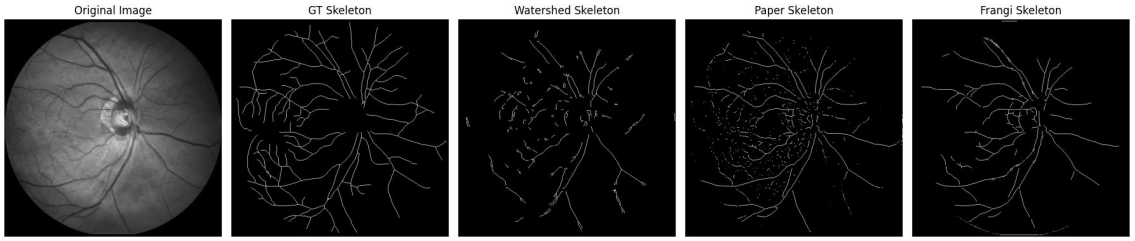

Figura 19: Squelettes pour l'image 7

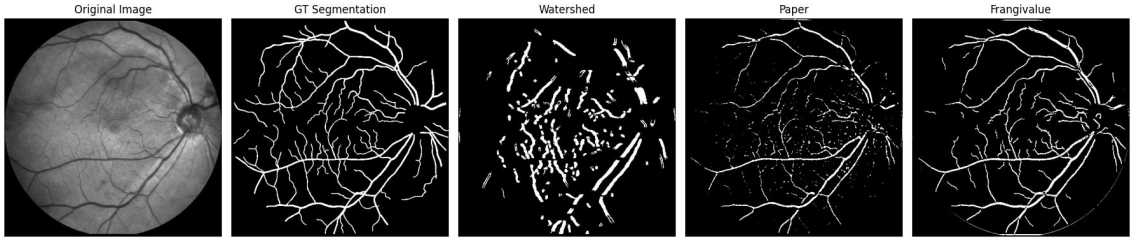

Figura 20: Segmentations pour l'image 8

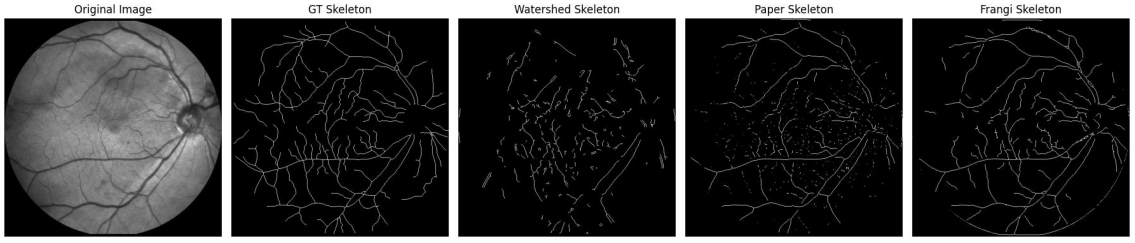

Figura 21: Squelettes pour l'image 8

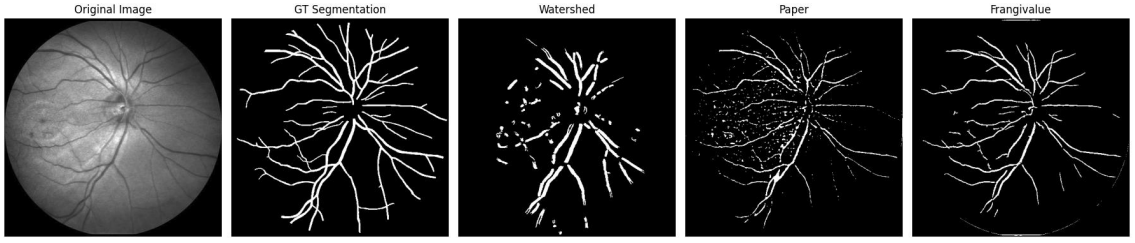

Figura 22: Segmentations pour l'image 9

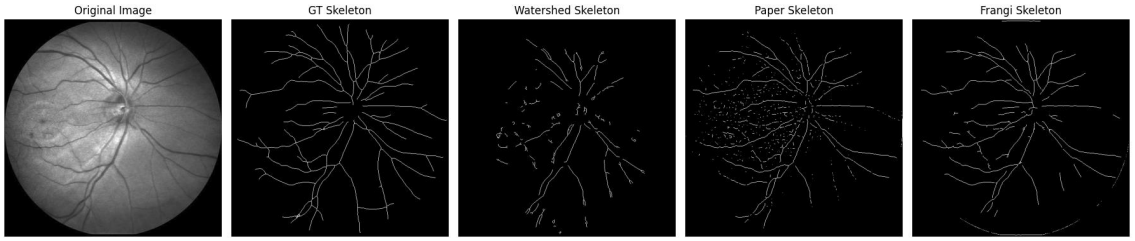

Figura 23: Squelettes pour l'image 9

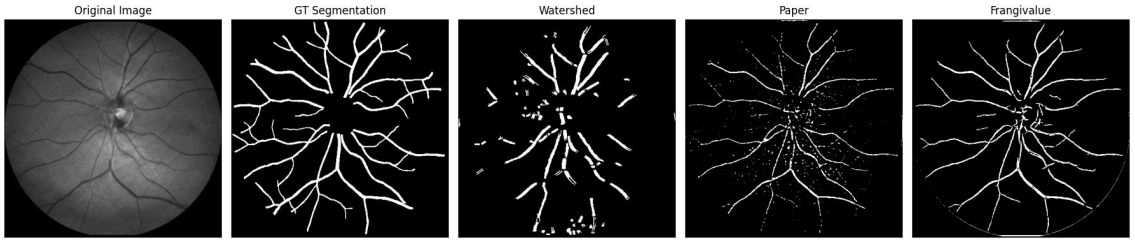

Figura 24: Segmentations pour l'image 10

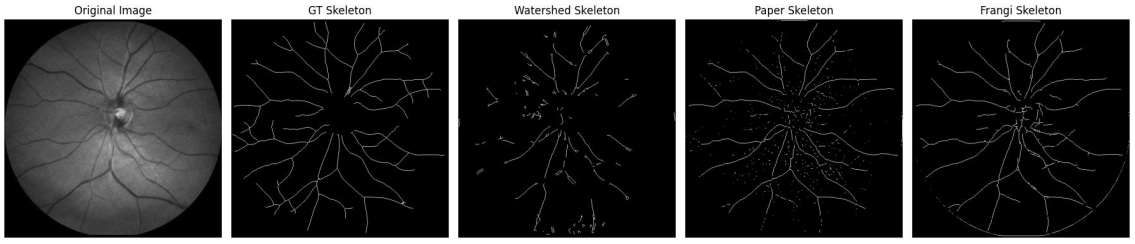

Figura 25: Squelettes pour l'image 10

3.2 Métriques

Pour mieux comparer les méthodes, on calcule quelques métriques pour chaque image. Elles sont les suivantes:

- **Précision** :

$$\text{Précision} = \frac{TP}{TP + FP}$$

  Elle mesure la proportion des vraies structures détectées parmi toutes celles détectées.

- **Rappel (Recall)** :

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

Il mesure la capacité à détecter toutes les vraies structures.

- **F1-score** :

$$F1 = 2 \cdot \frac{\text{Précision} \cdot \text{Rappel}}{\text{Précision} + \text{Rappel}}$$

C'est la moyenne harmonique entre la précision et le rappel.

On peut les voir dans les graphes ci-dessous:

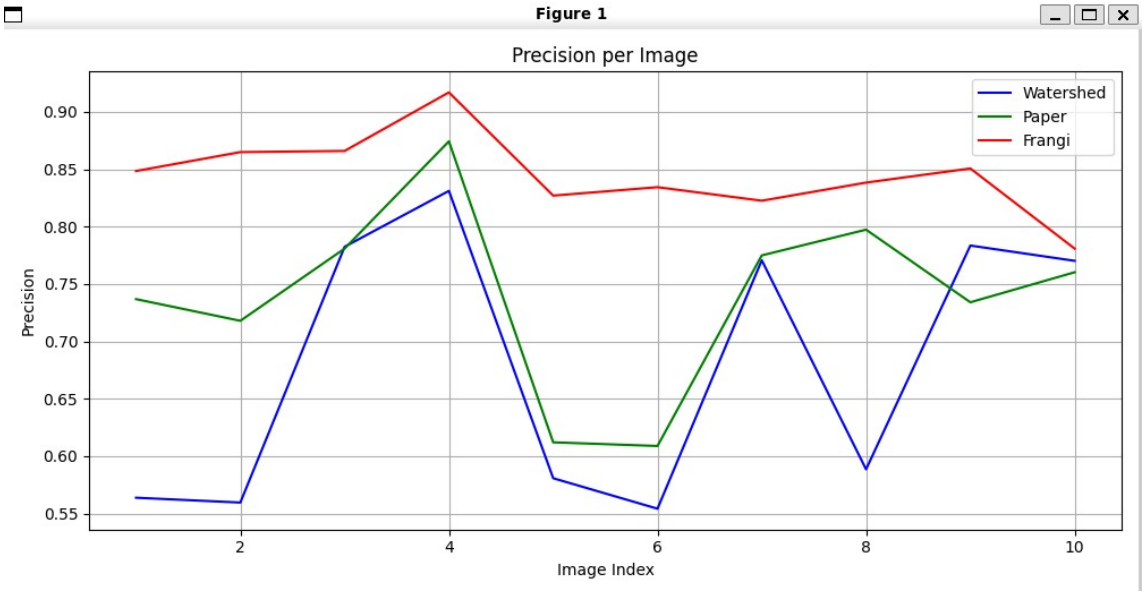

Figura 26: Précision des outputs pour chaque image et chaque méthode

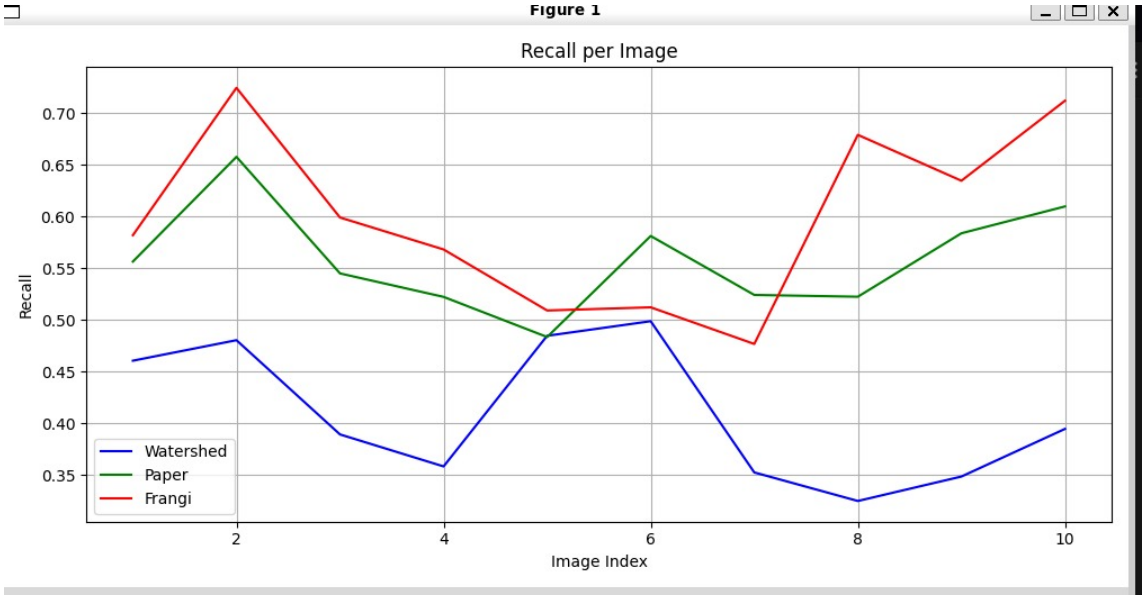

Figura 27: Recall des outputs pour chaque image et chaque méthode

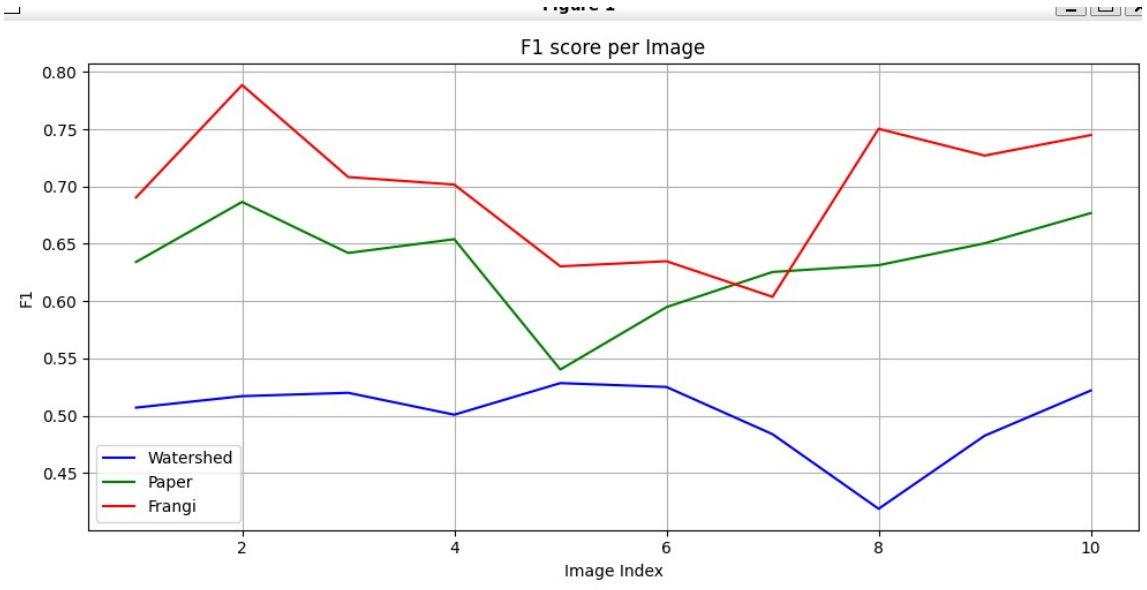

Figura 28: F1 score des outputs pour chaque image et chaque méthode

4 Conclusion

En conclusion, on remarque que la méthode la plus performante, visuellement et en termes de métriques, est la méthode de Frangi, car elle est très robuste pour éliminer le bruit et renforcer les vaisseaux.

Cependant, la méthode proposée dans l'article n'est pas mauvaise : elle obtient des résultats très similaires en ce qui concerne le F1-score. Peut-être que si la fusion des attributs avait été correctement implémentée, sa performance aurait été très proche de celle de Frangi, puisque la différence entre les deux méthodes réside principalement dans la détection des petites structures vasculaires et dans l'efficacité du débruitage.

En ce qui concerne la méthode Watershed, sa performance a été moyenne : elle détecte bien les gros vaisseaux, mais les plus petits sont souvent oubliés.

Pour conclure, ce travail a été très intéressant car il nous a permis de comprendre ce que chaque filtre et chaque opération morphologique est capable de faire sur une image. Il a surtout montré qu'il est possible de réaliser une segmentation sans avoir recours à un modèle d'intelligence artificielle récent, mais simplement en utilisant des techniques classiques et bien établies.

5 Bibliographie

Referências

FRANGI, Alejandro F. et al. Multiscale Vessel Enhancement Filtering. In: WELLS, William M.; COLCHESTER, Alan; DELP, Scott (Ed.). *Medical Image Computing and Computer-Assisted Intervention – MICCAI'98*. Springer Berlin Heidelberg, 1998. v. 1496. (Lecture Notes in Computer Science), p. 130–137. DOI: 10.1007/BFb0056195.

MANZANERA, Antoine. *Notes de cours – Compléments d'image*. fr. 2025. Cours magistral, Département d'informatique, ENSTA Paris. Disponible sur la plateforme Moodle.

ROSSANT, Florence et al. A Morphological Approach for Vessel Segmentation in Eye Fundus Images, with Quantitative Evaluation. *Journal of Medical Imaging and Health Informatics*, v. 1, p. 42–49, 2011. DOI: 10.1166/jmihi.2011.1006.