Projet final
Multi-Scale Line Detector
GBM8770 – Automne 2024

Professeure: Farida Cheriet
Chargés de laboratoire : Zacharie Legault et Emmanuelle Richer

---

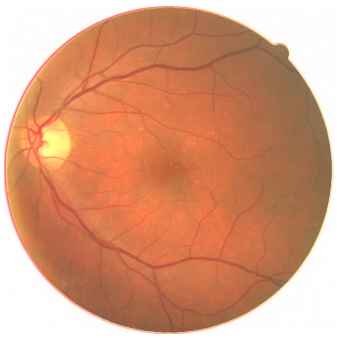
Objectifs : Dans un premier temps, ce projet vous amènera à implémenter un algorithme de segmentation et des outils d'analyse de cet algorithme. À cette occasion, vous allez évaluer la pertinence des métriques de segmentation proposées par l'article.

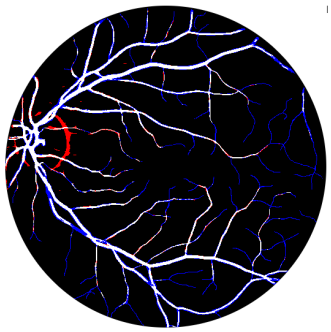
Dans un second temps, vous serez conduits à concevoir des expériences pour vérifier les hypothèses établies par les auteurs. À partir de ces expériences, vous devrez rédiger le résultat de vos expériences ainsi qu'une discussion.

Remise du travail : Ce travail est à réaliser en binôme et à remettre sur Moodle au plus tard le 5 décembre à 23h30. L'implémentation du projet doit être en Python. Le fichier de rendu doit être une archive ZIP nommée

`GBM8770_Projet_équipe_nom_matricule_nom_matricule`

(par exemple : `GBM8770_Projet_0_Legault_1234567_Richer_7654321`) contenant tous les fichiers nécessaires pour exécuter le code du projet.





> **i** Vous aurez peut-être besoin d'installer des nouveaux packages dans votre environnement **tpGBM** si ce n'a pas été fait dans les TPs précédents. Vous pouvez dans ce cas exécuter les commandes suivantes dans une **Anaconda Prompt**.

```
$ conda activate tpGBM
$ pip install opencv-python
$ pip install scikit-learn
```

Le squelette du code de la classe **MultiScaleLineDetector** et des fonctions utilitaires (notamment pour le chargement des données) vous sont fournis. Ces fichiers contiennent des commentaires **TODO** indiquant quelles parties du code sont à compléter et à quelles questions ils correspondent. Un certain nombre d'informations propres à l'implémentation Python sont décrites en commentaire du code, prenez soin de les lire !

Plus que pour les précédents laboratoires, portez votre attention sur la qualité du rapport (rédaction, pertinence de l'argumentation, etc.), en particulier pour les parties **Exercice III** et **Exercice IV**.

> **i** Vous avez à compléter des fonctions dans des fichiers **.py**, ainsi qu'un Jupypter Notebook comme vous le faisiez dans les anciens TPs. Ainsi, vous appelerez des fonctions codées dans les fichiers **.py** dans votre notebook. Pour aider au débuggage et vous aider à comprendre votre code et vos erreurs, vous pouvez travailler avec un éditeur de code tel que Visual Studio Code ou PyCharm. Vous pourrez ainsi débugger votre code ligne par ligne (en mettant des breakpoints) et vérifier les valeurs de vos variables. Vous n'êtes évidemment pas obligés, et vous pouvez également écrire et modifier vos fichiers **.py** depuis l'interface web Jupyter Notebook.

Exercice I Implémentation de la MSLD

Exercice I.1 Préparatifs

Pour entraîner et valider l'algorithme, nous allons utiliser la base de données DRIVE.

Q1. Complétez la fonction **load_dataset()** dans le fichier **dataset.py** pour qu'elle charge et renvoie la base de données d'entraînement et de test (en incluant pour chaque échantillon : l'image, le label et le masque). Puis affichez l'image, le label et le masque de la première

image de l'ensemble d'entraînement.

> **i** Assurez-vous que les **labels** et les **masques** sont bien en valeurs booléennes. Vous pouvez également utiliser la fonction **os.path.join** pour relier des chemins, et ce de façon sécuritaire sur tous les systèmes d'exploitation.

On notera cette image **I1** dans la suite de l'énoncé.

> **i** Chaque exemple d'entraînement et de test est stocké dans un objet de type **Sample**, qui contient les attributs **name**, **image**, **label**, et **mask**. Ces attributs sont accessibles avec la notation **sample.image**, **sample.label**, etc.

Exercice I.2 Basic Line Detector

Dans un souci d'optimisation de l'implémentation, nous allons utiliser des convolutions dès que cela est possible – particulièrement pour le calcul des moyennes d'intensités sur une fenêtre de taille **W** et le long des lignes de longueur **L**. De plus ces masques ne seront générés qu'une seule fois, lors de l'instanciation de la classe **MultiScaleLineDetector**.

Q1. Quel filtre de convolution permet le calcul de la valeur moyenne d'une fenêtre de taille **W** autour d'un pixel ? Complétez le constructeur de la classe **MultiScaleLineDetector** dans le fichier **algo.py** en définissant l'attribut **avg_mask**.

Q2. On souhaite implémenter le BLD en utilisant des masques de convolution. Ces masques viendront sélectionner les pixels le long d'une ligne. Ils seront définis par leur longueur **L** et une série d'orientations. Expliquez comment construire ces masques pour permettre le calcul de la valeur moyenne le long de lignes de taille **L** pour **num_orientations**. Complétez le constructeur de la classe **MultiScaleLineDetector** avec la définition de **line_detectors_masks**.

> **i** Afin d'instancier un objet de la classe **MultiScaleLineDetector**, on doit définir les attributs qui lui sont propres. Les attributs **self.line_detectors_masks, self.self.avg_mask** sont accessibles dans toutes les méthodes de **MultiScaleLineDetector**. On les utilisera dans les questions suivantes.

Vous pouvez à présent instancier l'objet **msld = MultiScaleLineDetector(W, L, num_orientations)** en remplaçant les hyperparamètres **W**, **L** et **num_orientations** par leurs valeurs conseillées par l'article.

Q3. À l'aide de la section 3.1 de l'article, implémentez `basic_line_detector(grey_lvl, L)`.

> ❗ Cette méthode devra retourner la carte de réponse `R` normalisée : $R' = \frac{R - R_{mean}}{R_{std}}$ (cf. équation 3 de l'article).

> ℹ Si besoin, allez voir les informations supplémentaires sur le BLD ainsi que la fonction `np.maximum`.

Q4. Affichez la réponse du filtre appliqué à l'image `I1` pour les longueurs `L=1` et `L=15`. Comparez les deux et commentez.

> ❗ Utilisez les arguments `vmin` et `vmax` pour imposer les bornes de l'intensité de vos images.

> ℹ Attention, le filtre BLD ne s'applique pas à tous les canaux de l'image. Relisez la section 3.1 de l'article pour plus d'informations.

Exercice I.3 Multi-Scale Line Detector

Q1. Implémentez `multi_scale_line_detector(image)` à l'aide des sections 3.2 et 3.3 de l'article.

Q2. Affichez le résultat de l'algorithme appliqué à l'image `I1`. Comparez avec les réponses du filtre BLD obtenues précédemment et commentez.

Exercice I.4 Apprentissage du seuil

Pour calculer le seuil donnant la meilleure précision sur l'ensemble d'entraînement efficacement, nous allons utiliser la courbe ROC (*Receiver Operating Characteristic*). Cette courbe sera étudiée en détail dans la partie **Exercice II**. Pour le moment, il vous faut juste savoir qu'elle associe à chaque seuil possible le taux de faux positifs et le taux de vrais positifs si ce seuil était choisi.

Q1. Donnez la formule de la précision en fonction du taux de faux positifs FPR, du taux de vrais positifs TPR, ainsi que du nombre de valeurs positives P, de valeurs négatives N et du nombre total de pixels S dans l'image de label.

Q2. Pour vous faire gagner du temps, on vous donne une fonction qui calcule le TPR et le

FPR associé à chaque seuil possible pour un dataset donné, `roc(msld, dataset)`. Assurez-vous que vous comprenez bien chaque ligne.

Pourquoi est-il si important de ne sélectionner uniquement les pixels qui appartiennent au masque ? En particulier, comment évoluerait la précision si on sélectionnait finalement tous les pixels de l'image (sachant que hors du masque la prédiction du MSLD est toujours nulle) ?

Q3. À l'aide de la fonction `roc(msld, dataset)` et de la formule de la question 1, complétez la fonction `learn_threshold(msld, dataset)` qui identifie le seuil pour laquelle la précision est la plus élevée.

Q4. Utilisez cette fonction pour apprendre le seuil sur les images d'entraînement. Pour quelle raison faut-il absolument conserver une partie des images (l'ensemble de test) et ne pas les utiliser pendant l'entraînement ?

> Il n'est pas anormal d'obtenir un seuil différent de celui annoncé par les auteurs.

Exercice I.5 Affichage et région d'intérêt

Q1. Implémentez la méthode `segment_vessels(image, threshold)` qui applique le seuil à la carte de réponse calculée sur une image. Affichez le résultat de l'algorithme sur l'image `I1`.

Q2. Vous devriez voir apparaître des erreurs de segmentation aux bords du fond d'oeil (à la frontière circulaire entre le fond d'oeil et le fond noir). À quoi est dû ce phénomène ?

Q3. Pour palier à ce problème, dupliquez les datasets, érodez la région d'intérêt (le masque) de manière à réduire le diamètre de 10 pixels, puis effectuez l'apprentissage du seuil à nouveau. Comparez les valeurs de seuil et la précision globale.

> En Python, les objets sont stockés par référence. L'instruction `train_copy = train` ne fait pas de copie en mémoire du dataset `train` et les deux variable pointent vers le même objet dans la mémoire. La modification de l'un modifiera l'autre. Utilisez plutôt :
>
> ```
> from copy import deepcopy
> train_copy = deepcopy(train)
> ```

Q4. On vous donne la fonction `show_diff(msld, sample, threshold, ax)` dans le fichier `viz.py` qui affiche les faux positifs en rouge, les faux négatifs en bleu, les vrais positifs en blanc et les vrais négatifs en noirs. Avec cette fonction, affichez la différence entre la prédiction et le

label sur l'image `I1`. Commentez le résultat.

> **i** Vous remarquerez que la fonction permet un argument optionnel `ax` qui a par défaut la valeur `None`. Si vous voulez afficher carte de différences avec d'autres subplots (créés avec `plt.subplots`), vous n'avez qu'à passer en argument le système d'axes dans lequel vous voulez faire l'affichage.
>
> ```
> fig, axs = plt.subplots(1, 3) # On a 3 subplots
> show_diff(..., ax=axs[0]) # On affiche la carte de différence dans le premier subplot
> ```

Exercice II Métriques de segmentation

Exercice II.1 Précision globale et locale

Q1. Implémentez la fonction `naive_metrics(msld, dataset, threshold)` qui évalue la précision et la matrice de confusion de l'algorithme pour un dataset donné. Calculez ces métriques sur l'ensemble de test.

Encore une fois n'oubliez pas de ne sélectionner que les pixels appartenant à la région d'intérêt !

> **!** Attention ! La matrice de confusion que retourne `naive_metrics` doit être normalisée par le nombre de labels positifs et négatifs.

Q2. Utilisez la fonction `show_diff` pour afficher la prédiction sur `I1` à différents seuils et calculez la précision associée à chacun de ces seuils. Indiquez la valeur du seuil et la précision dans le titre de chaque carte de différence. Commentez sur l'effet du seuil.

Q3. Les auteurs proposent une seconde métrique : la précision locale. Quelles raisons avancent-ils pour motiver cette proposition ?

Q4. Grâce à une opération morphologique, créez une copie du dataset `test` tel que son attribut `mask` corresponde à la région d'intérêt "locale" proposée par les auteurs. Puis, sans modifier la fonction `naive_metrics(msld, dataset, threshold)`, calculez la précision et la matrice de confusion locales.

Q5. Cette seconde métrique met plus en valeur l'algorithme proposé que la précision globale. Au vu des erreurs de l'algorithme révélées à la question 4 de l'**Exercice I**.5, donnez une raison supplémentaire (omise par les auteurs) qui explique ce phénomène.

Exercice II.2 Indice Dice (bonus)

De nombreux indices existent pour mieux évaluer les performances de segmentation que la précision globale : précision balancée, indice de Jaccard, kappa de Cohen, etc. Nous allons ici utiliser l'indice Sørensen-Dice. En notant Y les labels et $\hat{Y}$ les prédictions de l'algorithme, l'indice Dice est défini par

$$\mathrm{Dice}(Y, \hat{Y}) = 2\left(\frac{Y \cap \hat{Y}}{Y + \hat{Y}}\right).$$

Q1. En quoi cette métrique répond aux limitations de la précision globale ?

Q2. Implémentez la fonction **`dice(msld, dataset, threshold)`** et calculez sa valeur sur les régions d'intérêt globale et locale sur l'ensemble de test.

Exercice II.3 Courbe ROC et aire sous la courbe

Le choix d'un seuil de segmentation revient à faire un compromis entre faux-positifs et faux-négatifs (un seuil élevé limitera les faux-positifs mais augmentera les faux-négatifs et inversement). Le choix du bon compromis dépend bien souvent de l'application. Cependant les algorithmes se distinguent par le choix des caractéristiques à seuiller (ici un filtre MSLD) plutôt que par la méthode de sélection du seuil. La courbe ROC permet de représenter les performances de l'algorithme indépendamment du seuil choisi.

Pour construire cette courbe, on calcule le taux de faux-positifs (FPR) et le taux de vrais-positifs ($\mathrm{TPR} = 1 - \mathrm{FNR}$) pour chaque valeur de seuil. La courbe obtenue (FPR en abscisse, TPR en ordonnée) caractérise l'efficacité de l'algorithme à distinguer les vaisseaux du fond, indépendamment du seuil.

Enfin, pour simplifier la comparaison entre deux courbes ROC, on extrait leurs aires sous la courbe (AUC).

Q1. Que signifie une AUC de 1, de 0.5 ou de 0 pour les performances du modèle ?

Q2. Implémentez la méthode **`plot_roc(msld, dataset, ax)`** dans le fichier **`viz.py`** qui trace la courbe ROC et calcule son AUC. Puis faites de même pour la région d'intérêt globale et pour la région d'intérêt locale. Comparez les résultats et commentez.

Exercice III Validation de l'hypothèse de recherche de l'article

À l'aide de votre implémentation de l'algorithme et des différents cas du dataset de test, discutez de la qualité du protocole expérimental pour vérifier les hypothèses de recherche sous-jacentes de l'article.

Exercice IV Discussion

Q1. Discutez du choix des hyperparamètres `W`, `L` et `num_orientations`. Leurs valeurs ont-elles des justifications théoriques ? Quel est leur impact en pratique ?

Q2. Les auteurs ont choisi le seuil donnant *la meilleure précision* sur l'ensemble d'entraînement. D'après les analyses effectuées dans la partie **Exercice II**, discutez de ce choix.

Q3. En vous appuyant sur vos expérimentations pour les deux dernières parties de cet énoncé, proposez des recommandations pour améliorer l'algorithme.