

UNIVERSITE DE PARIS
UFR MATHÉMATIQUES ET INFORMATIQUE

**TER6 : Extraction et reconnaissance
de pointeurs pré-segmentés dans
des images médicales**

Master 1 Vision Machine Intelligente

Hocine Amine BENMEHREZ – Aïssatou SIGNATE

Encadré par Laurent Wendling

Année universitaire 2021 – 2022

Table Des Matières

1. INTRODUCTION	3
2. ETAT DE L'ART	4
3. SOLUTION LOGICIELLE	5
4. METHODES ETUDIEES.....	6
4.1 AU NIVEAU DU TRAITEMENT DES FLECHES	6
4.1.1 <i>Bounding Box</i>	6
4.1.2 <i>Méthode incrémentale</i>	8
4.2 AU NIVEAU DU TRAITEMENT DES ROI	14
4.2.1 <i>Affichage des ROIs</i>	14
4.2.2 <i>Extraction des ROIs</i>	15
4.2.3 <i>Traitement des ROIs</i>	16
5. CONCLUSION	18
6. BIBLIOGRAPHIE	18
7. ANNEXES	19
7.1 INSTALLATION DU PROGRAMME SUR ECLIPSE ET UTILISATION.....	19
7.2 AFFICHAGE DES ROIS TRAITEES	20
7.2.1 <i>Extraction des contours Sobel</i>	20
7.2.2 <i>Filtre médian</i>	21
7.2.4 <i>Egalisation histogramme</i>	22

1. Introduction

L'imagerie médicale est un domaine de la médecine regroupant des techniques permettant aux professionnels d'acquérir et de restituer les parties du corps humain. Ces outils d'analyse permettent aux médecins de réaliser des diagnostics (blessures et maladies) tout en évitant d'utiliser des moyens invasifs. Ce domaine a connu de grands progrès grâce à l'informatique, en particulier avec le traitement numérique d'image.

L'informatique intervient en tant qu'aide à la décision de plusieurs façons : en automatisant des opérations, en réduisant le bruit sur les images acquises, augmenter la netteté des images... Le but étant de garantir au mieux l'information portée par ces images médicales.

Un des outils mis à la disposition des professionnels est le marqueur. Celui-ci permet d'indiquer une zone d'intérêt pouvant indiquer une anomalie et suivre leur évolution. Toutefois, il existe plusieurs types d'anomalies détectables sur les images médicales : ceci peut aller de la fracture d'un os sur une radio, à la découverte d'une infection suite à une IRM. Ces images constituent une grande base de données, il est nécessaire de réaliser la classification de ces pathologies de façon automatique.

Nous avons à notre disposition, une base de données constituée de plusieurs centaines d'images médicales présentant des flèches (pointeurs/marqueurs) pré-segmentées. Les flèches ont pu être détectées et classifiées grâce aux travaux de recherches réalisées les années précédentes. Ainsi une partie de l'objectif initial a été atteint à savoir l'extraction des régions et puis de les classer dans deux classes (pointeurs et autres). Nous avons décidé de nous focaliser sur l'extraction des régions d'intérêt (ROI) pointées par les flèches. En ciblant des ROIs détectées par des professionnels, nous pouvons procéder à des techniques de traitement d'image permettant d'améliorer le rendu de la zone, et de transmettre une meilleure qualité d'information, tout ceci dans l'espoir d'obtenir des diagnostics plus précis.

Dans ce rapport, nous présenterons notre approche des extractions et du traitement de ces ROI. Dans un premier temps, nous avons focalisé notre étude sur la détection de l'orientation de la flèche dans l'image. Une fois trouvée, il nous suffit de déterminer la taille de la zone à cibler, de l'extraire, et puis de la traiter avec différentes techniques de traitement numériques.

2. Etat de l'art

La détection des flèches est une première étape clé de l'étiquetage ROI et de l'analyse du contenu des images. Il existe actuellement des méthodes dédiées à la reconnaissance des flèches sur des images médicales.

Élaborée en 2015 par un groupe de chercheurs issues de différentes universités ¹, une technique basée sur la signature géométrique permet de détecter les annotations de flèches sur les images biomédicales. Les images sont d'abord binarisées à l'aide d'un outil de binarisation floue (fuzzy logic), afin d'en extraire des composantes connexes (CC) qui seront considérées comme étant des candidates, c'est-à-dire des flèches potentielles.

Pour chaque candidate, des propriétés géométriques sont vérifiées afin d'établir leurs signatures. Elles sont par la suite comparées avec des signatures théoriques, qui sont issues de flèches connues (vérité terrain).

Un score de similarité important permet de classer un candidat en tant que flèche. Elles présentent une précision de 93.14% et un rappel de 86.12%, surpassant les méthodes de détection de flèches précédemment rapportées. Les données ont été évaluées par rapport à la collection de référence imageCLEFmed.

Une autre méthode a été élaborée en 2021 par un groupe d'étudiant en master issu de l'université de Paris Cité. Ils sont passés par une étape d'extraction des CC de l'image en binarisant l'image à l'aide de fonctions fournis par la bibliothèque graphique OpenCV. Par la suite, ils appliquent la Generic Fourier Descriptor comme descripteur de forme, et calculent un vecteur caractéristique pour chaque CC. Ils ont décidé d'utiliser ce descripteur en raison de ses différents avantages : invariant à la rotation, à la translation et au changement d'échelle.

Pour classer l'ensemble de ces CC, deux approches de classification ont été proposées : supervisée et non supervisée.

La première, compare les vecteurs caractéristiques des CC inconnues avec celles de CC déjà connues et établies en tant que flèche (vérité terrain constituée de 6 flèches). La

¹ K. Santosh, L. Wendling, S. Antani et G. R. Thoma, «Overlaid Arrow Detection for Labeling Regions of Interest in Biomedical Images,» *IEEE*, 2016.

comparaison est réalisée à l'aide de l'indice de similarité de Jaccard, avec un seuil de similarité décidé (0.68). Cet indice permet de calculer la similarité entre 2 ensembles. Cette approche a obtenu un taux de précision de 72%, et de rappel de 55%.

La seconde approche utilise un algorithme de clustering, appelé K-means. Selon le nombre k choisi en entrée, nous obtenons k clusters. Ici, les clusters vont comporter des CC calculées grâce à leurs vecteurs caractéristiques. Par la suite, l'ensemble des flèches seront classifiées dans les différentes classes établis par cette méthode. Cependant, le grand nombre de CC présentes dans l'image qui ne sont pas des pointeurs rend le clustering plus difficile. Finalement, c'est l'approche supervisée qui a été retenue.

3. Solution Logicielle

Dans le cadre de ce projet, notre encadrant nous a suggéré d'utiliser le code élaboré l'année précédente en tant que « boîte noire ». Nous avons apporté certaines modifications à celui-ci afin d'obtenir les données nécessaires à l'extraction et traitement des ROIs. L'ensemble de ces éléments se trouve dans un dossier intitulé stock2 : il comporte les images des CC reconnues en tant que flèche. Ces images sont rognées, sous la forme d'un rectangle (boîte englobante) depuis l'image originale. Le nom de ces miniatures correspond alors aux coordonnées de ces boîtes englobantes.

Concernant notre code, nous avons décidé d'utiliser le langage de programmation Python, ainsi que l'IDE PyCharm. Python propose de nombreuses bibliothèques dédiées au traitement de l'image : Pillow /PIL (Python Imaging Library), NumPy et Matplotlib (pour l'affichage des différents résultats obtenus lors du traitement des flèches)

Nous avons également utilisé la bibliothèque graphique Open CV afin d'appliquer les différents types de traitements sur les ROI extraites (filtre médian, filtre passe-haut, filtre de Sobel et égalisation de l'histogramme).

Pour lancer les codes sur Eclipse et PyCharm veuillez-vous référer à l'annexe 7.1.

4. Méthodes étudiées

Dans cette partie, nous aborderons les différentes méthodes appliquées sur les CC reconnues en tant que flèche par le système élaboré l'année précédente. Nous vous présenterons également des méthodes étudiées sur les ROIs détectées.

4.1 Au niveau du traitement des flèches

Dans un premier lieu nous avons décidé de focaliser notre étude sur 6 types de flèches afin d'élaborer dans notre vérité terrain.

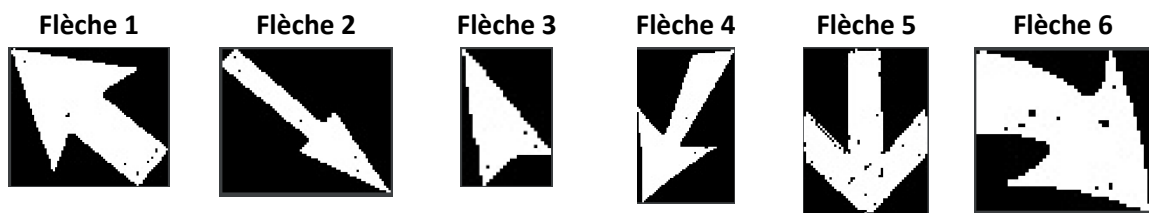


Figure 1 Flèches constituant la vérité terrain

D'après les travaux réalisés les années précédentes, ces flèches semblent apparaître le plus fréquemment dans la base d'image. Ces images étant bruitées, il nous a été suggéré, dans un premier temps de les supprimer de façon manuelle. Par la suite, nous appliquerons différents filtres de lissage pour réduire ce bruit dans notre système de façon automatique.

Afin d'obtenir l'orientation de la flèche nous procédons en deux étapes : déterminer l'axe directeur de la flèche, puis sa direction.

4.1.1 Bounding-Box

Les flèches étant des formes géométriques, plus précisément des polygones concaves il est possible de les recouvrir d'une enveloppe convexe. Cette étape permet de dessiner une zone approximative, dans notre cas rectangulaire, autour d'une image binaire. Open CV propose une méthode **cv2.boundingRect()**, qui met en évidence une région d'intérêt. Elle est appliquée après une extraction d'une carte des contours réalisée à l'aide de la fonction **cv2.findContours**.

Bien que les images sur lesquelles nous nous basons semblent présenter uniquement une flèche, cette étape de pré-processing réduit la zone de traitement, et donc réduit les risques de confusions et d'erreurs entre le fond et la forme de l'image.

Une fois la bounding-box créée, nous calculons le squelette de la flèche afin de déterminer des points de terminaisons. Ces points permettront de distinguer la tête et la queue de la flèche. Pour cela, il suffit d'appliquer un produit de convolution avec la matrice suivante :

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 10 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

Ce filtre identifie les points d'extrémités d'un pixel de large dans image binaire. Nous utilisons la méthode `cv2.filter2D` d'Open CV afin de réaliser le produit de convolution entre l'image et le filtre.



Figure 3 Squelette de la flèche 1



Figure 2 Points d'extrémités de la flèche 1

Nous supposons que les points représentent la tête et la queue de la flèche, nous devons alors les classifier

Sur cette image, nous pouvons considérer ces pixels comme étant des éléments d'un nuage de points (figure3). Nous pouvons faire un K-means (Open CV propose une méthode) pour grouper les pixels en deux clusters ($k=2$) : tête et queue.

Sur les images suivantes, le point rouge représente la tête de la flèche, et le bleu la queue. Pour obtenir un axe directeur, nous avons décidé de relier les deux points par un segment de couleur verte.

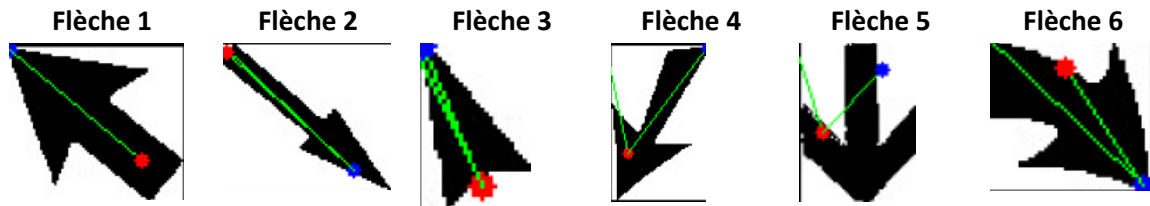


Figure 4 : Résultat sur les 6 flèches

Parmi les 6 flèches nous avons pu déterminer correctement l'axe directeur de 3 flèches uniquement, et la bonne direction pour seulement la flèche 4. Nous avons décidé par la suite d'abandonner cette approche en raison de ses faibles résultats.

En effet, nous avons élaboré une autre méthode présentant de meilleurs résultats. Nous allons vous l'expliquer dans la partie suivante.

4.1.2 Méthode incrémentale

Un premier fichier appelé **methode_généralisée.py** a été créé en se basant sur les 6 flèches vu précédemment (vérité terrain). Cette méthode consiste à traiter les pixels un à un et de stocker dans une liste les coordonnées des pixels ayant la même couleur que les contours de la flèche traitée. Ces pixels sont situés sur l'image de la flèche (que nous appellerons miniature). Nous procédons ensuite à un calcul de distance entre chaque point de cette liste afin de déterminer la base et la pointe de la flèche.

Nous avons également exécuté ce programme sur un plus grand nombre et une plus grande variété de flèches afin de déterminer son efficacité. Au fur et à mesure, le code fut amélioré, notamment grâce au fichier **Flèche.py** qui a complémentarisé **methode_généralisée.py**. Nous avons pris en considération les nouvelles caractéristiques des nouvelles flèches : la vérité terrain s'est donc élargit à 53 flèches.

Déroulement du programme **methode_généralisée.py** :

Preprocessing : Les flèches extraites ne peuvent pas être utilisées directement dans notre programme. En effet, les miniatures doivent d'abord passer par étape de binarisation. Ceci est réalisé à l'aide d'une méthode proposée par le module **PIL**.

Un filtre médian de taille 3 est appliqué afin de diminuer le bruit impulsionnel (type poivre et sel) tout en gardant les extrémités de la flèche (comme la pointe) le plus net possible.

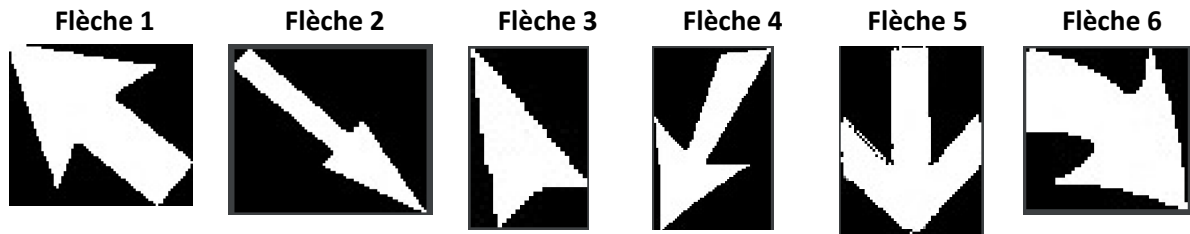


Figure 5 Flèches après réduction du bruit

Déterminer la couleur de la flèche : Dans l'optique de rendre notre méthode plus performante, nous avons constaté que déterminer la couleur de la flèche était essentiel pour obtenir des résultats.

Nous avons tenté diverses approches telles que :

- La comparaison entre le nombre de pixels blancs et noirs trouvés dans l'image
- La comparaison des valeurs de pixels sur les 4 coins de la miniature.
- Décider de la couleur selon la valeur du pixel du milieu de la miniature.

Finalement, nous avons décidé de comparer la valeur des pixels situés sur le cadre de la miniature : la première ligne et colonne de pixels ainsi que la dernière ligne et colonne de pixels.

La couleur dominante sur le cadre correspondra alors à la couleur du fond tandis que la couleur restante sera celle de la flèche.



Figure 6 Exemple de de couleur de fond et de formes possible

Extraction des coordonnées de la flèche située sur le cadre de la miniature : Une fois la couleur de la flèche déterminée, nous stockons dans une liste toutes les coordonnées des pixels de cette couleur et situés sur le cadre : ce sont des points d'extrémités.

Dans le cas où aucun pixel de la valeur du contour de la flèche est détecté sur une ligne ou colonne du cadre, nous élargissons la zone de détection sur la ligne et colonne voisine.

Recherche de la base : En observant les 6 images de base (cf. figure 4), nous avons remarqué qu'en considérant uniquement les pixels de la flèche situé sur le cadre de la miniature, la distance entre les 2 points extrêmes qui forment la base ont souvent la plus courte distance. Nous avons donc décidé de se baser sur cette observation pour trouver la base de la flèche traitée.

Nous avons utilisé la distance euclidienne :

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Cependant, lors du calcul des distances, le programme prend en considération les pixels voisins comme ayant la distance la plus courte comme base de la flèche.

Pour pallier ce problème, nous avons décidé d'appliquer un seuil de 7 pixels sur la distance minimum acceptable entre 2 points. Ce seuil atteint, ces derniers pourront être considérés comme étant la base de la flèche. Cette valeur a été choisi après plusieurs Trials and Errors sur les 6 images de flèches.

Une fois ces 2 points extrêmes trouvés, le programme calcul le point situé au milieu du segment de droite créé entre ces 2 points en utilisant la formule ci-dessous :

$$Point\ milieu = \left(\frac{x_1 + x_2}{2}, \frac{y_1 + y_2}{2} \right)$$

Recherche de la pointe : Tout comme l'étape précédente, nous calculons les distances mais cette fois-ci, entre la base trouvée et les différents points trouvés sur le cadre de l'image. Le point réalisant la distance maximum avec la base sera considéré comme la pointe de notre flèche.

Affichage de la direction : Open CV propose la méthode **cv.arrowedline** afin de dessiner une flèche de la couleur que l'on souhaite sur notre miniature. Nous avons rentré les coordonnées de la base et de la pointe trouvée en paramètre.

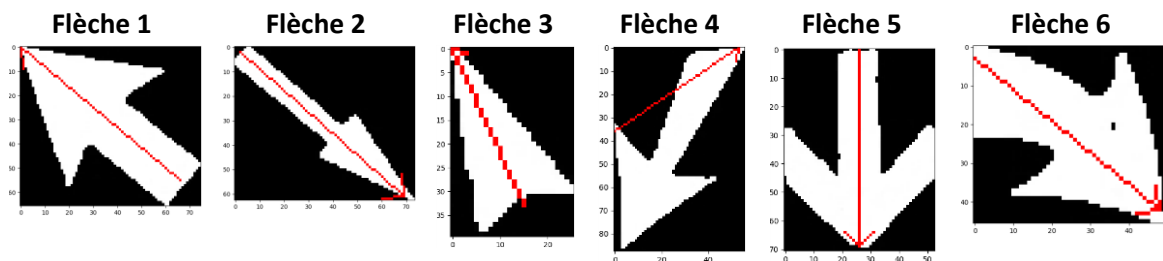


Figure 7 Affichage de la direction trouvée par methode_généralisée

Tableau 1 Résultat sur 6 miniatures

	Nombre de flèches
Bonne direction	5
Mauvaise direction	1
Total	6
Taux de réussite	83%

Le programme a été testé sur une plus large base de données de flèche afin de tester son efficacité sur d'autres types de flèches. Nous avons sélectionné des images de flèches parmi les composantes connexes détectés par l'algorithme des étudiants de l'année passée afin de compléter notre base de données de flèches.

Quelques exemples de résultats parmi les 53 miniatures testées

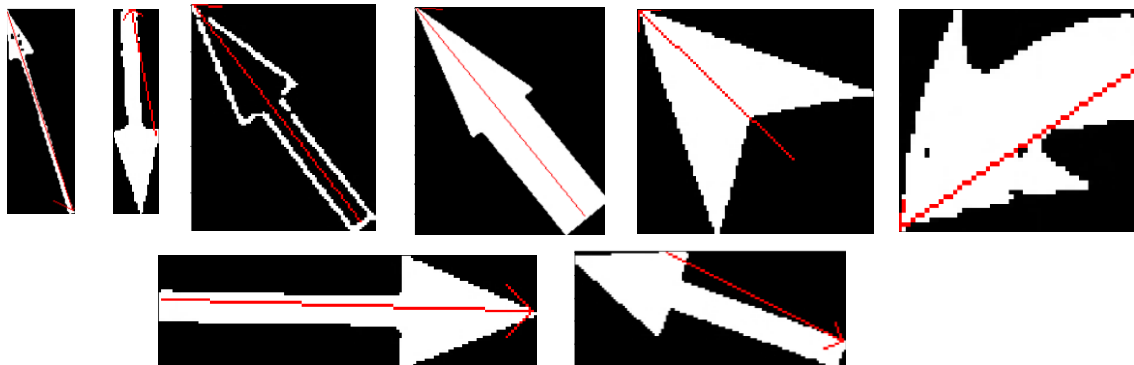


Figure 8 direction trouvée par methode_généralisée.py sur de nouvelles images de flèches

Tableau 2 Résultat sur 53 images

	Nombre de flèches
Bonne direction	30
Mauvaise direction	23
Total	53
Taux de réussite	56.6%

Ce tableau ci-dessus représente les résultats de l'exécution du programme **methode_généralisée.py** seul. Comme nous pouvons le constater, le taux de réussite a considérablement baissé lorsqu'on a testé notre programme sur 53 images (56.6%) en comparaison avec 6 images seulement (83%).

Nous avons donc décidé de réaliser un programme complémentaire nommé **Flèche.py**. Il fut élaboré en prenant en considération davantage de flèches différentes.

Déroulement du programme Flèche avec **methode_généralisée** :

Dans cette partie, nous allons expliquer le déroulement des différentes étapes de Flèche ainsi que les différences avec **methode_généralisée**.

Preprocessing et recherche de la couleur du contour : le programme utilise les mêmes fonctions que celle de **methode_généralisée.py** pour améliorer l'image et trouver la couleur de la flèche.

Extraction des points de la flèche sur le cadre de l'image : Contrairement à **méthode_generalisée.py** qui stocke dans une liste tous les pixels de la flèche présente sur le cadre de la miniature, ici nous stockons uniquement 2 points de chaque côté du cadre.

Ces 2 points représentent les premiers pixels rencontrés sur le cadre de la miniature en commençant par le début puis la fin de chaque ligne et colonne.

Une fois ces 2 points extraits, nous calculons le milieu entre les 2 points de chaque côté du cadre afin de ne garder que 4 points que nous stockerons dans une liste.

Détection du coin de l'image ayant la même valeur que la couleur de la flèche : Le programme Flèche est exécuté différemment suivant la taille de la liste retournée.

- **2 coins ou plus détectés** : Si le programme trouve au moins 2 coins de l'image ayant la même valeur que celle des contours de la flèche, le programme **methode_généralisée** sera exécuté.



Figure 9 Image de flèche avec au moins 2 coins détectés

- **1 coin détecté :** Le programme retournera une liste avec 3 coordonnées (les coordonnées du coin, ainsi que les 2 points du cadre qui ne forment pas ce coin) puis calculera la distance entre ces 3 points avec le même principe utilisé pour **methode_généralisée.py** : les points ayant la distance minimum trouvée seront ceux utilisés pour trouver les coordonnées de la base de la flèche. Puis nous calculons la distance entre ce dernier point avec le reste des points afin de trouver la pointe de la flèche qui réalisera une distance maximum.



Figure 10 Image d'une flèche avec 1 coin détecté

- **Aucun coin détecté :** Notre programme nous retourne alors une liste de coordonnées de 4 points, un point pour chaque côté du cadre. Nous calculons les distances entre ces 4 points afin de trouver d'abord les 2 points de la base qui nous permettront de calculer la base de la flèche. Nous avons 2 possibilités de déroulement du programme :
 - ❖ Les 2 points qui forment la base ont été extraits des côtés du cadre qui se font face (côté droit avec le gauche ou coté haut avec le bas du cadre): dans ce cas, nous calculons la distance entre la base trouvée et les 2 points qui n'ont pas été utilisés pour calculer la base de la flèche. La pointe sera le point qui réalisera une distance minimum avec la base de la flèche.



Figure 11 Image de flèche avec 2 points qui forment la base ont été extraits de côté du cadre qui se font face

- ❖ Si la première condition n'est pas remplie, nous effectuons un calcul de distance entre la base de la flèche et chaque point du cadre afin de déterminer la pointe de la flèche qui réalise une distance maximum avec la base de la flèche.

Une fois la base de la flèche et la pointe sont détectées, nous pouvons déterminer la direction de cette flèche en calculant l'axe directeur à partir de ces 2 points.

4.2 Au niveau du traitement des ROI

4.2.1 Affichage des ROIs

Les zones d'intérêts sont indiquées par les flèches c'est pour cela que nous devons d'abord trouver la direction de la flèche à l'aide de sa base et de sa pointe.

Pour afficher ces ROIs, nous suivons les étapes suivantes :

- **Trouver les coordonnées de la flèche dans l'image d'origine** : Chaque miniature a pour nom ses coordonnées obtenues depuis l'image d'origine. Il nous reste plus qu'à effectuer un split de son nom. Ceci permet de trouver le point de l'image de la flèche avec la coordonnée (0,0).
- **Trouver les coordonnées de la pointe et de la base de la flèche dans l'image d'origine** : Pour y parvenir, nous additionnons les coordonnées du coin en haut à gauche de l'image extraite par l'étape précédente avec les coordonnées de la pointe et de la base de la flèche.
- **Entourer les ROIs** : Nous mettons en évidence les ROIs en les entourant d'un cercle. Le rayon de ce cercle est déterminé en calculant la distance entre la base et la pointe de la flèche. Puis nous calculons les coordonnées de son centre qui se trouve à une distance de la pointe égale à la distance entre la base et la pointe.

$$X_{centre} = 2 * X_{pointe} - X_{base} + X_{image_origine}$$

$$Y_{centre} = 2 * Y_{pointe} - Y_{base} + Y_{image_origine}$$

- **Afficher les ROIs** : Maintenant que nous avons les coordonnées du centre de notre cercle et son rayon, nous utilisons une méthode proposée par Open CV, **cv2.circle**, qui permet d'entourer ces zones d'intérêts.

Nous répétons les mêmes actions pour chaque flèche détectée afin d'avoir l'ensemble des ROIs de l'image indiqués par ces flèches.

Nous enregistrons une image sous le nom de **ROI_cercles.png** qui affiche toute les ROIs encerclés dans notre image d'origine.

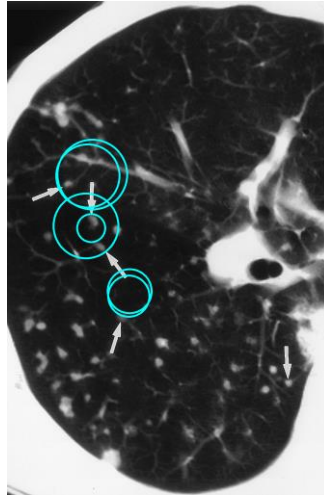


Figure 12 L'image ROI_cercles.png

4.2.2 Extraction des ROIs

Ayant les coordonnées du centre de chaque ROI, nous décidons d'extraire ces zones d'intérêts sous forme de rectangle. Pour cela, nous avons besoin de la ligne de départ et d'arrivée ainsi que les colonnes de départ et d'arrivée. Elles sont trouvées en additionnant et en soustrayant aux coordonnées du centre la valeur du rayon afin de déterminer la ligne et colonne de départ ainsi que la ligne et colonne d'arrivée respectivement.



Figure 13 ROI extraite de l'image d'origine

Nous créons une image à l'aide d'une méthode d'Open CV **cv.imwrite** qui va afficher toutes les ROIs trouvées dans l'image d'origine en remplaçant les valeurs des pixels de la nouvelle image par celle du rectangle représentant notre ROI.

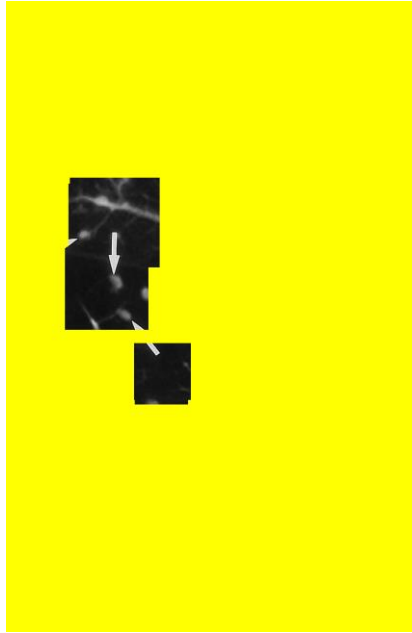


Figure 14 Image de toute les ROIs extraites (sans traitement)

Nous décidons de mettre le fond en jaune afin de mieux voir les résultats.

4.2.3 Traitement des ROIs

Une fois l'ensemble des ROIs extrait sur l'image médicale d'origine, nous procédons à quatre types de traitement :

- Egalisation de l'histogramme.
- Deux filtres passe-haut qui permettent tout deux d'améliorer la netteté de l'image mais avec des rendus différents :
 - Un filtre passe haut qui permet d'extraire les contours d'une image.
 - Un filtre de Sobel, qui permet d'extraire les contours. Une méthode est disponible sur Open CV.
- Un filtre passe bas : nous utiliserons un filtre médian.

Voici un exemple d'une série de traitement appliquée sur une ROI :



Figure 15 ROI égalisée





 <p>Figure 16 Égalisation</p>	<p><u>Egalisation de l'histogramme :</u> Grâce à cet ajustement, les intensités peuvent être mieux réparties sur l'histogramme. Cela permet aux zones de contraste local plus faible d'obtenir un contraste plus élevé.</p>
 <p>Figure 17 Filtre passe haut</p>	<p><u>Filtre passe haut :</u> Ce traitement permet d'accentuer les contours et les détails des images.</p>
 <p>Figure 18 Filtre Sobel</p>	<p><u>Filtre de Sobel :</u> Le filtre de Sobel est également un filtre passe haut. Ce filtre est relativement simple à utiliser, et est plus efficace en termes de temps.</p>
 <p>Figure 19 Filtre médian</p>	<p><u>Filtre médian :</u> Ce type de filtre permet de préserver les contrastes, et de diminuer le bruit impulsionnel (type poivre et sel) qui peut être présent sur des images dégradées. Le filtre médian permet d'obtenir de bons résultats pour filtrer ce type de bruit.</p>

Tableau 3 : Ensemble des traitements proposés sur une ROI

Vous trouverez en annexe l'affichage final des ROIs depuis une image médicale.

5. Conclusion

Nous avons pu extraire une partie des ROIs pointées par des professionnels de santé sur une imagerie médicale à l'aide de flèches. Nous avons appliqué dessus quelques traitements d'imagerie numériques pour améliorer leur contenu. Afin d'obtenir ces résultats nous nous sommes basés sur l'étude réalisée l'année précédente par un groupe d'étudiants issu de l'Université de Paris Cité. Ces derniers ont pu extraire une partie des CC (considérées comme flèches) présentes sur les images médicales. Par la suite, nous avons déterminé l'orientation des flèches : celles-ci s'orientaient vers une zone d'intérêt. Une fois la zone trouvée, nous l'avons extraite et pratiqué les traitements suivants : égalisation de l'histogramme, filtre passe-haut et filtre passe-bas. Ceci afin de présenter sous différentes formes une même ROI.

Bien que nous ne puissions pas diagnostiquer et interpréter ces ROIs sur le plan médical, nous espérons que ce type de traitement permettra aux professionnels de santé d'obtenir une meilleure qualité d'information et les aider dans la prise de décision.

Il est possible d'améliorer la détection des ROIs en rendant meilleur le système de classification élaboré l'année précédente : ceci aura pour conséquence direct d'augmenter le nombre de ROI pouvant être traitées.

Un autre axe d'amélioration serait d'adapter le choix du traitement d'image selon le type d'imagerie médicale que nous avons : radiologie, scanner ou IRM. Ceci pourrait constituer la première étape vers la classification des types de pathologies ciblés par ces flèches.

6. Bibliographie

[1] K. Santosh, L. Wendling, S. Antani et G. R. Thoma, «Overlaid Arrow Detection for Labeling Regions of Interest in Biomedical Images,» *IEEE*, 2016.

7. Annexes

7.1 Installation du programme sur Eclipse et utilisation

Pour la partie Eclipse et détection des flèches :

https://docs.opencv.org/master/d1/d0a/tutorial_java_eclipse.html

Tout d'abord, notre programme se lance sur eclipse, il faut donc l'installer si vous ne l'avez pas déjà à cette adresse : <https://www.eclipse.org/downloads/>

Nous utilisons également la bibliothèque Open CV, vous pouvez donc le télécharger à cette adresse : <https://opencv.org/releases/>

Pour installer Open CV sur eclipse commencez par ouvrir eclipse puis cliquez sur « Window » -> « Preferences ».

Sur la fenêtre des préférences, cliquez sur « Java » -> « Build Path » -> « User Libraries » puis cliquez sur le bouton « New » sur la droite de la fenêtre.

Ecrivez « OpenCV » ou le nom que vous voulez pour le nom de la library.

Cliquez sur la library que vous venez de créer puis cliquez sur « Add External JARs »

Parcourez votre dossier d'installation d'OpenCV jusqu'à « /build/java » puis sélectionnez « opencv-(votre-version).jar

Enfin Cliquez sur « Native Library Location » puis « Edit puis sélectionnez dans le dossier d'installation d'OpenCV le dossier « /build/java/x64 » ou x32 si votre machine est 32-bit

Maintenant que OpenCV est installé sur eclipse vous pouvez importer notre projet en cliquant sur « File » -> « Import » -> « General » -> « Existing Projects Into Workspace ».

Cliquez sur « browse » puis sélectionnez le dossier où vous avez extrait notre projet puis « Finish »

Notre projet devrait à présent apparaître à gauche dans le liste, naviguez jusqu'à la classe « Test » puis lancez le programme à partir de là.

Vous pouvez tester le programme avec d'autres images que celles fournies en changeant celles situées dans le dossier « ref » du projet. ²

^{2 2} Issue du rapport rédigé sur l'étude réalisé l'année précédente.

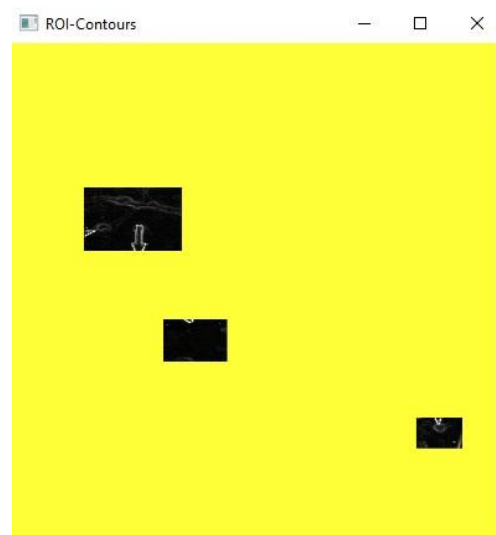
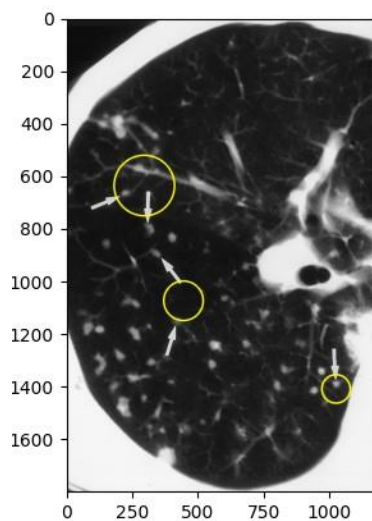
Pour la partie Python :

A la fin de l'exécution de ce programme (sur Eclipse), un dossier intitulé stock2 apparaîtra. Ce dossier comporte l'ensemble des CC considérées en tant que flèches. Nous nous sommes basées sur ces éléments pour déterminer les ROIs. Il suffira de déplacer ou copier ce dossier dans le dossier du code fourni cette année.

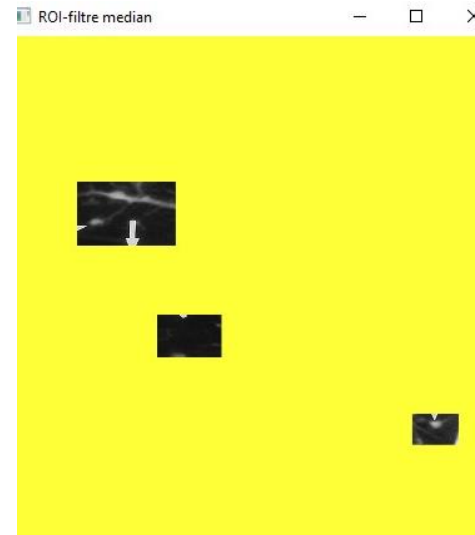
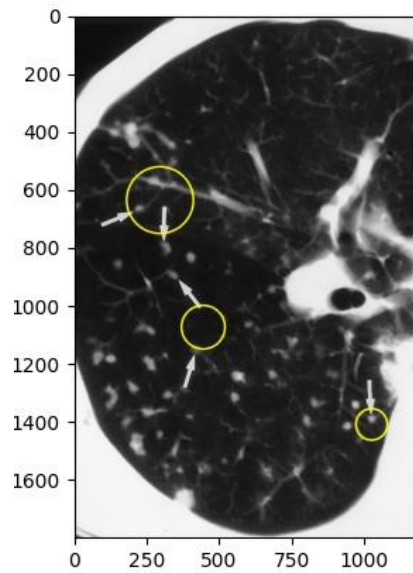
En ce qui concerne la partie élaborée en Python, nous avons utilisé l'IDE PyCharm, mais vous pouvez utiliser l'IDE de votre choix. Les bibliothèques suivantes devront être installées au préalable : Pillow, NumPy, Matplotlib et Open CV.

7.2 Affichage des ROIs traitées

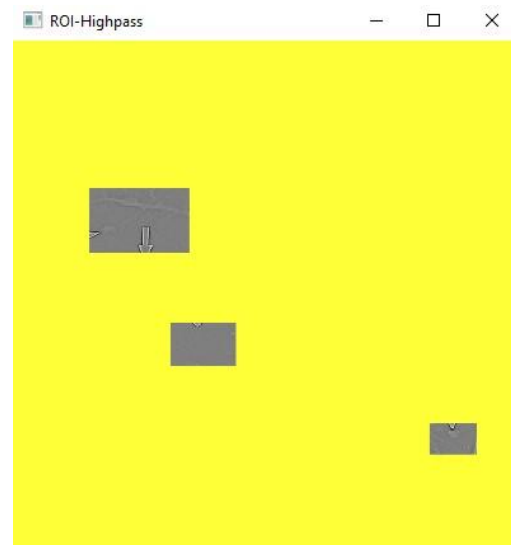
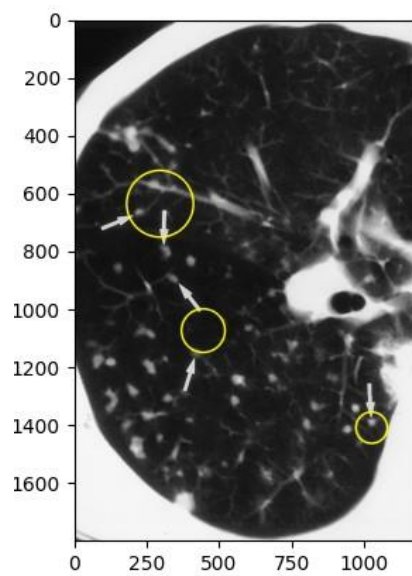
7.2.1 Extraction des contours Sobel



7.2.2 Filtre médian



7.2.3 Filtre passe haut



7.2.4 Egalisation histogramme

