Rapport de projet TER

Extraction automatique de pointeurs similaires dans des images médicales

Benjamin KAUFFMANN et Frédéric IENG

Encadré par : Laurent WENDLING

Année 2020-2021

Table des matières

Résumé

Pour ce projet, nous nous intéressons à l'extraction des pointeurs dans des images médicales. Les praticiens utilisent souvent ces pointeurs pour désigner des zones d'intérêt (ROI pour *Region Of Interest*) pouvant être associées à différentes pathologies.

Les bases d'images s'élargissent de plus en plus, il devient donc intéressant de trouver un moyen de recenser automatiquement les zones sélectionnés par les pointeurs afin de mieux catégoriser les pathologies et leurs évolutions.

Il existe des modèles robustes d'extraction de pointeurs mais il est nécessaire de construire une vérité terrain au préalable. Les vérités terrains sont généralement crées en mettant des étiquettes manuellement sur des pointeurs.

Nous modélisons une nouvelle approche de classification non supervisée pour obtenir de manière automatique la localisation des pointeurs sans mis en place de vérité terrain.

Dans notre approche, nous utilisons un *descripteur de Fourier générique* (GFD) et l'algorithme du *K-mean++* pour rassembler les éléments similaires dans une même classe. Nous affinons ensuite nos résultats en filtrant nos éléments avec des critères géométriques.

1 Introduction

Ce projet tutoré s'inscrit dans le cadre du Master **Vision et Machine Intelligente** (VMI) à l'*Université de Paris* (anciennement *Paris Descartes*). Ce projet a pour objectifs de nous familiariser avec les outils utilisés pour le traitement d'image et la reconnaissance des formes ainsi que modéliser une nouvelle approche de la classification non supervisée afin de comprendre les avantages et les limites de cette approche.

2 Description du projet

2.1 Consigne du projet

Notre objectif est de modéliser une nouvelle approche de classification non supervisée permettant d'obtenir la localisation de ces marqueurs contenues dans des images médicales binaires.

Pour cela nous passons par plusieurs étapes :

- **Extraction des composantes connexes** contenues dans l'image binaire,
- **Élimination des composantes** de taille trop grande, trop petite, non compacts ou collées au bord (seuils à définir),
- **Calcul d'un descripteur de Fourier générique** (GFD) sur chaque composante,
- **Modélisation/ Création d'une approche de clustering** pour déterminer les k classes de composantes les plus pertinentes,
- **Extraction des nouvelles couches** associé à chaque image,
- Suivant l'état d'avancement, une étude de l'axe directeur et des extrémités des pointeurs pourra être réalisée pour **déterminer la zone d'intérêt à segmenter** par la suite.

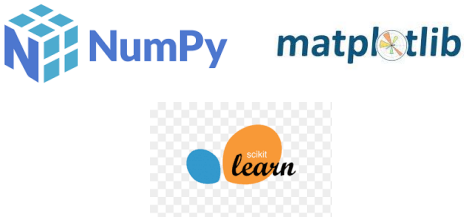
2.2 Technologies utilisées

Nous avons choisi de programmer en **Python** pour notre projet, car c'est un langage de programmation beaucoup utilisé pour créer des modèles d'apprentissage automatique (*Machine Learning*). Nous pouvons ainsi créer facilement nos modèles de classification non supervisés.

Il existe des bibliothèques (*Libraries*) de **Python** assez populaire pour le traitement de don-

nées. Nous utilisons donc ces bibliothèques pour faciliter notre tâche :

- **Numpy** qui permet d'effectuer des calculs matriciels de manière optimale. Une image peut être modéliser comme une matrice, nous traiterons nos images plus rapidement avec cette bibliothèque.
- **Matplotlib** qui permet de visualiser nos données.
- **Scikit-learn** qui permet de créer des modèles de classification facilement.



Pour lisons et traitons nos images avec la bibliothèque *OpenCV* qui permet de traiter nos images efficacement.

Pour collaborer efficacement et travailler sur la même version du projet, nous utilisons les outils **Git** et **GitHub** pour la gestion de version.

Pour exporter facilement notre projet sur d'autres machines, nous utilisons le logiciel de conteneurisation, **Docker**.

3 État de l'art

Il existe des méthodes de détection de pointeur utilisant les propriétés géométriques des pointeurs. Nous pouvons notamment citer les travaux de M.Wendling dans les articles suivants, "**Overlaid arrow detection for labeling biomedical image regions**" et "**A New Way to Detect Arrows in Line Drawings**". Dans ces articles, les pointeurs sont définis comme l'union de deux figures géométriques, un triangle isocèle T (mais non équilatéral) et un rectangle R relié à la base du triangle. On notera que la bissectrice du sommet principal de T est également une médiane du rectangle R.

Tout d'abord, M.Wendling et ses collègues ont effectué une binarisation hiérarchique floue afin de mettre en valeur les pointeurs qui peuvent parfois être flous. Puis ils extraient des composantes connexes qui sont des candidates pour l'extraction de pointeurs.

Pour chaque candidats, ils cherchent les points clés de ces composantes en étudiant les résultats de quatre balayages sur le cadre orthogonal. Ils effectuent un balayage horizontal et vertical à partir des coins en haut à gauche et en bas à droite. Une fois les points clé trouvés, ils ont utilisés les propriétés géométrique des pointeurs pour filtrer les composantes candidates.

Pour commencer, ils cherchent l'axe de symétrie du pointeur qui correspond à la bissectrice du sommet principal du triangle T. Il peut exister plusieurs points candidat pouvant représenter le sommet principal, C du triangle T. Ils commencent donc par estimer l'axe de symétrie, Sym(C) en étudiant la distribution des points autour des axes candidats avec la formule suivante :

$$Sym(X_i) = \sup_{t \in \left[1,\ldots,\frac{p}{2}\right]} \left\{ \frac{\sum_{i=1}^{p} min\left(V_{X_i}^{t+i}, V_{X_i}^{t-1}\right)}{\sum_{i=1}^{p} max\left(V_{X_i}^{V_{t+i}}, V_{X_i}^{t-1}\right)} \right\},$$

$$Sym(C) = \underset{i=1,\ldots,n}{argmax}\{Sym(X_i)\}.$$

Une fois l'axe de symétrie estimé, ils cherchent les deux autres points du triangle T qu'ils nommèrent A et B avec la formule suivante :

Cardinality

$$= \frac{min(|V_A|, |V_B|)}{max(|V_A|, |V_B|)}, \{V_A^i, V_B^i, t_\pi(V_C^i)\} \simeq 1.$$

A partir des points A,B et C trouvé, ils calculent l'aire théorique du candidat avec la formule de Héron. Cette formule permet de calculer l'aire d'un triangle à partir des longueurs des segments qui la compose. Ils ont ensuite comparé le résultat de ce calcul avec l'aire réelle du candidat. L'aire calculé avec la formule de Héron et l'aire réelle du candidat devraient être proche si les points A,B et C qu'ils ont trouvé représentent bien les sommets d'un triangle.

$$Area = 1 - \left(\frac{K - H}{H}\right),$$

$$H = \sqrt{(l(l-a)(l-b)(l-c))},$$

$$l = 1/2(a + b + c).$$

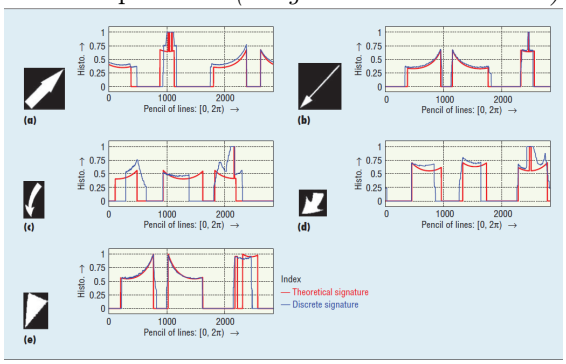
Dans l'article "**Overlaid arrow detection for labeling biomedical image regions**", ils calculent et comparent à partir des points A, B et C, la signature discrète et la signature théorique du candidat pour vérifier le chevauchement de la composante candidate avec les éléments de l'arrière plan. Plus il y a de chevauchement, moins elle a de chance de représenter un pointeur.

FIGURE 1 – Taux de reconnaissance des pointeurs *(image venant de l'article 1)*

Reco

$$= SR \times \left(1 - \frac{\sum_{i=1}^{p} \sup\begin{pmatrix} \min\left(V_A^i, V_B^i\right), \min\left(V_A^i, t_\pi\left(V_C^i\right)\right), \\ \min\left(V_B^i, t_\pi\left(V_C^i\right)\right) \end{pmatrix}}{\Sigma_{i=1}^{p} \sup\left\{V_A^i, V_B^i, t_\pi\left(V_C^i\right)\right\}}\right).$$

FIGURE 2 – Signature théorique et discrète de différents pointeurs *(image venant de l'article 1)*



Ils suppriment ensuite les candidats qui apparaissent plusieurs fois dans leur jeu de données et comparent avec leurs résultats obtenues avec les résultats obtenues à partir d'autre méthodes notamment les résultats qu'ils obtenues avec des modèles de pointeurs prédéfinis. La méthode développée par M.Wendling et ses collèges semblent donner les meilleurs résultats.

Dans l'article "**A New Way to Detect Arrows in Line Drawings**", M.Wendling et M.Tabonne utilisent cinq critères pour détecter les pointeurs.

FIGURE 3 – Liste de critères pris en compte*(image venant de l'article 2)*

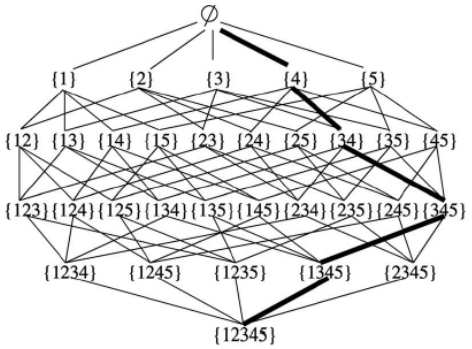
Criteria	Formula
$C1$	$Sym = \max_{t\in[1,p/2]}\left\{\frac{\sum_{i=1,p}\min(V^{t+i},V^{t-i})}{\sum_{i=1,p}\max(V^{t+i},V^{t-i})}\right\}$
$C2$	$Card = \min(\lvert V_A\rvert, \lvert V_B\rvert)/\max(\lvert V_A\rvert, \lvert V_B\rvert)$
$C3$	$\bar{O}ver = 1 - \frac{\sum_{i=1,p}\inf\{V_A^i,V_B^i,t_\pi(V_C^i)\} - 2\cdot d(A,B)}{\sum_{i=1,p}\sup\{V_A^i,V_B^i,t_\pi(V_C^i)\}}$

FIGURE 4 – Suite des critères pris en compte*(image venant de l'article 2)*

Criteria	Formula
$C4$	$Area = 1 - (K - H)/H$
$C5$	$Proto = D(V, \bar{S})$

Ils créent ensuite un treillis de mesures floues à partir des cinq critères et calculent l'importance de chaque critère en utilisant l'intégrale de *Choquet* et les indices de *Shapley* pour calculer l'importance de chaque critère.

FIGURE 5 – Treillis de mesures floues *(image venant de l'article 2)*



Lattice of the Fuzzy Measure

FIGURE 6 – Poids des chemins du treillis *(image venant de l'article 2)*

v(∅) = 0
V(4) = 0.038524
v(34) = 0.400977
v(345= 0.601545
v(1345) = 0.800386
v(12345) = 1

FIGURE 7 – Importance de chaque critères *(image venant de l'article 2)*

Criterion	*Sym*	*Card*	*Over*	*Area*	*Proto*
Shapley	0.72	0.51	**1.03**	0.76	**1.97**

La méthode développé par M.Wendling et M.Tabonne donnent des résultats robustes pour la détection de flèches mais nécessitent la mise en place d'un vérité terrain pour cette méthode.

FIGURE 8 – Vérité terrain utilisé par M.Wendling et M.Tabonne *(image venant de l'article 2)*

SERIES	SYMBOLS	y
1		1.
2		0.8
3		0.2
4		0.2
5		0.

4 Jeux de données et prétraitement

4.1 Présentation du jeu de donnée

Nous utilisons un ensemble d'image fourni par M. Wendling. Nous travaillons sur des images médicales binaires présentant des pointeurs de formes et tailles variables. Les pointeurs peuvent avoir une couleur différente selon l'arrière-plan.

FIGURE 9 – Exemple d'image médicale avec des pointeurs blancs et des pointeurs noirs

FIGURE 10 – Exemple d'image médicale avec des pointeurs noirs sur un arrière-plan noir *(peu visible mais bien présent)*

FIGURE 11 – Exemple d'image médicale avec des pointeurs blancs sur un arrière-plan noir

4.2 Traitement de l'image

Avant de classifier et extraire nos composantes connexes nous avons besoin d'obtenir les contours de notre image. Cependant les plupart des filtres permettant l'extraction de contours sont particulièrement sensibles aux bruits de l'image.

Nous utilisons donc le filtre de Canny qui est plus robuste et efficace sur des images bruitées. Le filtre de Canny se déroule en plusieurs étape, tout d'abord nous commençons par atténuer le bruit en utilisant un filtre de débruitage.

Nous avons évité les filtres médian et moyenneur car ils dégradent généralement les contours qui sont importants dans notre approche. Nous appliquons donc un flou Gaussien (*Gaussian Blur*) qui semble être le filtre de débruitage le plus adapté. En effet, il est efficace et dégrade moins les contours.

Pour appliquer le flou Gaussien, nous calculons le produit de convolution entre l'image et un noyau associé (*Figure 12*). Dans notre cas nous utilisons un flou de Gauss 5×5.

FIGURE 12 – Noyau associé à un flou de Gauss 5×5

$$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

Une fois le bruit atténué, nous appliquons le filtre de Sobel sur l'image pour obtenir l'intensité et la direction des contours.

FIGURE 13 – Noyaux utilisé pour calculer l'intensité des contours

$$G_x = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}; G_y = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$

FIGURE 14 – Valeur du gradient (*l'intensité du contour*)

$$|G| = \sqrt{G_x^2 + G_y^2}$$

FIGURE 15 – Direction du gradient

$$\theta = \arctan \frac{G_y}{G_x}$$

Sur l'image filtrée qu'on a obtenu, une plus forte intensité indique une probabilité plus haute de la présence d'un contour. Nous supprimons les points qui ne correspondent pas à des maxima locaux afin de conserver uniquement les points qui ont une haute probabilité de représenter un contour,

Enfin, nous effectuons un seuillage à hystérésis, pour cela nous fixons un seuil haut et un seuil bas. Nous conservons ensuite les composantes connexes qui contiennent au moins un point au dessus du seuil haut et supprimons tous les points en dessous du seuil bas. Nous obtenons ainsi les contours de l'image.

4.3 Extraction des composantes connexes

À partir de ces contours, nous extrayons un ensemble de composantes connexes. Pour cela nous utilisons une technique appelé *"Border Following"* pour effectuer une analyse de la structure topologique de l'image binaire. Ce procédé est décrit en détail dans l'article 4.

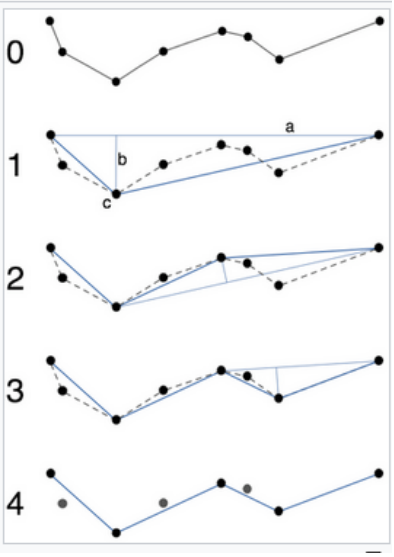
La technique permet de tracer la frontière entre les composantes connexes et les éléments de l'arrière-plan. Les points formant la frontière des différentes composantes connexes auront une étiquette différente, nous pourrons ainsi obtenir la liste des composantes connexes.

Nous effectuons ensuite une approximation polygonale de nos composantes connexes en utilisant l'algorithme de Douglas-Peucker. L'algorithme est récursif et utilise le principe de "*diviser pour régner*".

Initialement on sélectionne les deux points les plus éloignés du polygone. Ce sont les bornes. Pour chaque étape, on parcourt tous les points entre les bornes et on choisit le point le plus éloigné du segment formé par les bornes :

1. S'il n'y a aucun point entre les deux bornes, l'algorithme se termine la.
2. Si la distance entre le point le plus éloigné et le segment est inférieur à un certain seuil, on supprime tous les points entre les bornes.
3. Si la distance est supérieur au seuil, on divise le polygone en deux sous-parties : de la première borne au points sélectionné et du points sélectionné à la borne finale. Nous appliquons ensuite l'algorithme à nos deux sous-parties et fusionnons le résultat.

FIGURE 16 – Résultat de l'algorithme de Douglas-Peucker sur une courbe (*image de Wikipedia*)



Nous obtenons ainsi une approximation polygonale pour chacun de nos composantes connexes. Nous utilisons cette approximation par la suite pour filtrer les éléments qui ne nous intéressent pas.

5 Méthodes et évaluation

5.1 Critères géométriques

Afin d'éviter de traiter les éléments qui ont que peu de chance de représenter des pointeurs, nous filtrons les composantes connexes obtenues. Nous commençons par éliminer les composantes qui ont une aire trop grande ou trop petite car les pointeurs ont une taille raisonnable. Nous n'avons pas trouvé de moyen de déterminer automatiquement les seuils pour filtrer les composantes connexes selon l'aire, nous avons donc déterminer nos seuils manuellement de manière empirique.

Pour la suite, nous nous inspirons des travaux de M.Wendling qui sont résumés dans les articles, "**Overlaid arrow detection for labeling biomedical image regions**" et "**A New Way to Detect Arrows in Line Drawings**".

Nous utilisons les propriétés géométriques des pointeurs pour filtrer les composantes qui ne nous intéressent pas. Nous considérerons qu'il y a deux types de pointeurs : des pointeurs composé uniquement d'un triangle isocèle et des pointeurs composé d'un triangle triangle isocèle et d'un rectangle relié à la base du triangle.

FIGURE 17 – Image médicale contenant les deux types de pointeurs

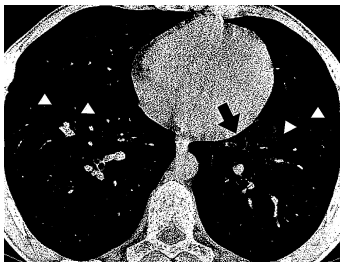


FIGURE 18 – Pointeur composé d'un triangle et d'un rectangle

FIGURE 19 – Pointeur composé uniquement d'un triangle isocèle

Nous filtrons les éléments qui n'appartiennent à aucune des deux catégories. Nous remarquons que les pointeurs ont soit sept sommets pour les pointeurs composé d'un triangle et d'un rectangle, soit trois sommets pour les pointeurs composé uniquement d'un triangle.

Nous conservons donc uniquement les composantes connexes dont l'approximation polygonale donne des polygones qui ont un nombre de cotés proche de trois ou sept.

Parmi les éléments candidats pour représenter des pointeurs à trois cotés , nous conservons uniquement ceux qui ont une aire proche de l'aire du triangle formé par les points de l'approximation polygonale. Nous calculons l'aire théorique du polygone avec la formule de Héron ci-dessous.

$$H = \sqrt{(l(l-a)(l-b)(l-c))},$$

$$l = 1/2(a+b+c).$$

avec a, b et c représentant chacun la longueur d'un segment de l'approximation polygonale. Nous conservons uniquement les éléments dont l'aire réelle est compris entre 0.75 et 1.25 fois l'aire théorique.

Sur ces éléments candidats filtrés, nous gardons uniquement les éléments les plus susceptibles de représenter un triangle isocèle. Pour cela, nous calculons la longueur des segments et si deux segments ont une longueur assez proche, nous considérons que les trois points forment un triangle isocèle.

5.2 Calcul d'un descripteur de Fourier générique (GFD)

Nous calculons ensuite le descripteur de Fourier générique sur les composantes connexes qui restent. L'algorithme qui permet de calculer le descripteur de Fourier générique peut se résumer en quatre étapes principales :

1. Normaliser la translation
2. Calculer la transformée de Fourier polaire
3. Normaliser la rotation
4. Normaliser l'échelle

Nous utilisons l'algorithme décrit dans l'article 3 dans notre cas. Voici les détails de chaque étapes :

1. Normaliser la translation :
   (a) Calculer la taille de l'image $f(x, y)$;
   (b) Calculer le centre de l'image de coordonnées (x_C, y_C) ;
   (c) Définir ce centre comme l'origine de l'image ;
2. Calculer la transformée de Fourier polaire avec FR la partie réelle de la transformée et FI la partie imaginaire de la transformée :
   (a) Calculer le rayon maximum de l'image ($maxRad$) ;
   (b) Soit rad la fréquence radiale, m la fréquence radiale maximale, ang la fréquence angulaire et n la fréquence angulaire maximum.

   Pour rad de 0 à m,
   pour ang de 0 à n,
   pour x de 0 à la largeur de l'image,
   et pour y de 0 à la hauteur de l'image :
   {
   $radius = \sqrt{(x - x_C)^2 + (y - y_C)^2}$
   $\theta = \arctan 2[(y - x_C)/(x - y_C)]$;
   Si $(\theta < 0) : \theta \mathrel{+}= 2\pi$;
   $FR[rad][ang] \mathrel{+}= f(x, y) \times \cos[2\pi \times rad \times (radius/maxRad) + ang \times \theta]$;
   $FI[rad][ang] \mathrel{-}= f(x, y) \times \sin[2\pi \times rad \times (radius/maxRad) + ang \times \theta]$;
   }
3. Normaliser la rotation et l'échelle et calcul du GFD :
   — Pour rad de 0 à m et ang de 0 à n :
   — Si $(rad = 0 et ang = 0)$:
   $DC = \sqrt{(2 \times FR^2[0][0])}$;
   $GFD[0] = DC/(\pi \times maxRad^2)$;

   — Sinon :
   $GFD[rad \times n + ang] = \sqrt{(FR^2[rad][ang] + FI^2[rad][ang])}/DC$;

Nous appliquons cet algorithme aux composantes connexes qui restent et obtenons ainsi le descripteur de Fourier générique pour chaque composante connexe.

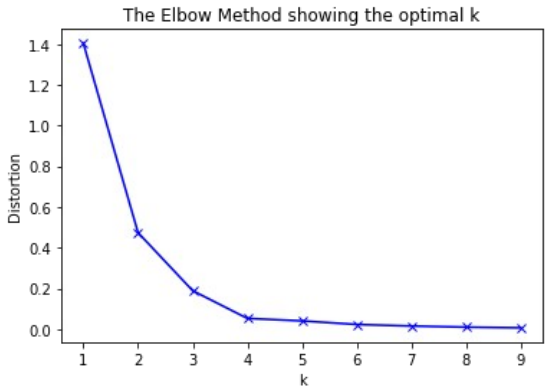
5.3 Mise en place du *K-mean++*

Avant de classifier, nous déterminons le nombre de classe optimal à l'aide de la méthode du coude (*Elbow Method*).

Pour cela, nous calculons l'inertie interclasse qui représente la somme des distances entre les centres de gravité des différentes classes. Nous cherchons le nombre de classe pour lequel l'inertie interclasse ne baisse plus assez.

Dans l'exemple ci-dessous (*Figure 20*), en appliquant la méthode du coude, le nombre idéal de classe semble être de trois ou quatre. Nous avons déterminé que le nombre optimal de classe était de trois dans notre cas.

FIGURE 20 – L'inertie interclasse en fonction du nombre de classe (*image venant de Wikipedia*)



L'algorithme du *K-means* permet de minimiser la distance entre les points à l'intérieur de chaque classe. Pour cela, l'algorithme va prendre des points aléatoires dans le jeu de données qu'il définira comme centres initiaux. Le nombre de centre initial que nous choisissons à été déterminé à l'avance, pour cela nous pouvons utiliser la méthode du coude. Ensuite, pour chaque itération jusqu'à ce que le résultat converge, on déroulera ces étapes :

1. Assigner à chaque point une classe selon le centre le plus proche
2. Pour chaque classe, assigner un nouveau centre qui est la moyenne des points qui la compose

FIGURE 21 – Algorithme du *K-means*(image venant de *Wikipedia*)

L'algorithme converge en général très rapidement, mais il est possible qu'il n'y arrive pas. Nous mettons donc en place un nombre maximum d'itération pour éviter les boucles infinies.

Nous avons choisi de classer nos différentes composantes connexes en utilisant l'algorithme du *K-means++*. C'est une variante de l'algorithme du *K-means*.

Nous préférons l'algorithme du *K-means++* car contrairement à l'algorithme du *K-means*, nous pouvons reproduire nos expériences en utilisant les mêmes paramètres, ce qui nous permet de comparer nos résultats plus facilement.

La différence se situe au niveau de la sélection des points initiaux, au lieu d'être complètement aléatoire à chaque fois, nous choisissons une graine (*seed*) qui donne les mêmes résultats dans les mêmes conditions.

6 Expérimentations

6.1 Exemple

Nous commençons par lire toutes les images d'un fichier. Nous utilisons les fonctions de la bibliothèque *OpenCV* pour traiter les images. Nous montrons le résultat de nos traitements sur la figure (*Figure 22*) ci-dessous.

FIGURE 22 – Image sur lequel nous travaillons

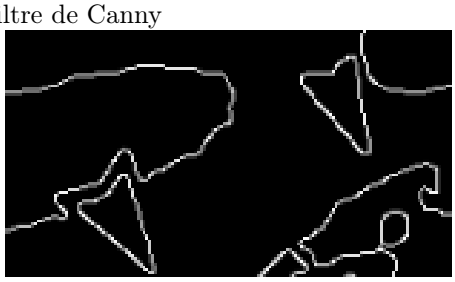
Sur chaque image, nous atténuons le bruit avec un flou Gaussien avec la fonction *GaussianBlur*.

FIGURE 23 – Deux pointeurs avant le flou Gaussien

FIGURE 24 – Deux pointeurs après le flou Gaussien

Nous appliquons ensuite le filtre de Canny pour obtenir les contours de notre image. Pour cela, nous utilisons la fonction *Canny*.

FIGURE 25 – Deux pointeurs après l'application du filtre de Canny



Nous récupérons ensuite des composantes connexes avec la fonction *findContours* de la bibliothèque d'*OpenCV*. Nous effectuons une approximation polygonale avec la fonction *approxPolyDP*, et filtrons les composantes connexes qui ne correspondent à nos critères géométriques.

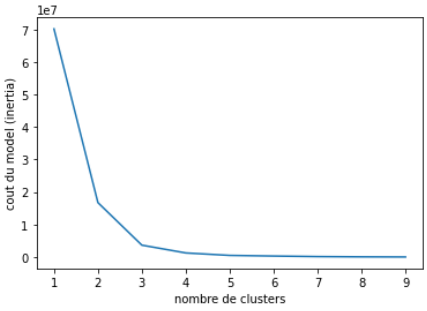
FIGURE 26 – Deux pointeurs après avoir filtré avec les critères géométriques

FIGURE 27 – Approximation polygonale des deux pointeurs

FIGURE 28 – Approximation polygonale des deux pointeurs avec les sommets encerclé

Nous utilisons ensuite la méthode du coude, pour déterminer le nombre de classes optimal. Pour cela, nous calculons l'inertie interclasse.

FIGURE 29 – Méthode du coude sur l'image de la *Figure 22*



Nous remarquons que l'inertie baisse beaucoup moins après trois classes, nous classons donc les

éléments dans trois classes.

Nous calculons ensuite le descripteur de Fourier générique qui nous permettra de classer nos différentes composantes connexes. Nous utilisons ensuite les fonctions de la bibliothèque *Scikit-learn* pour effectuer la classification non supervisé.

Nous créons ensuite un modèle avec *KMeans* et l'entraînons avec la fonction *fit_predict*. Dans la figure ci-dessous (*Figure 30*), chaque élément avec une couleur différente correspond à une classe différente.

FIGURE 30 – Résultat de la classification non supervisée

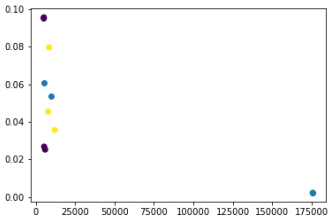


FIGURE 31 – Exemple d'éléments classés dans la classe 1

FIGURE 32 – Un élément classé dans la classe 2

FIGURE 33 – Exemple d'éléments classés dans la classe 3

Nous remarquons que la plupart des pointeurs ont été retrouvés dans une des trois classes, notre méthode semble viable.

6.2 Les limites de notre approche

Tout d'abord, nous remarquons qu'il est assez difficile d'évaluer l'efficacité de notre méthode. En effet, nous n'avons pas de vérité terrain qui permettrait d'évaluer de manière automatique si les éléments que nous trouvons représentent bien des pointeurs, nous ne pouvons pas utiliser des métriques comme le score F1 comme pour la classification supervisée.

Nous remarquons également que parfois certains pointeurs sont inclus dans composantes connexes plus grandes lorsqu'il y a contact entre ces deux éléments.

FIGURE 34 – Deux pointeurs qui seront "absorbés"

FIGURE 35 – Les deux pointeurs sont "absorbés" par une composante connexes

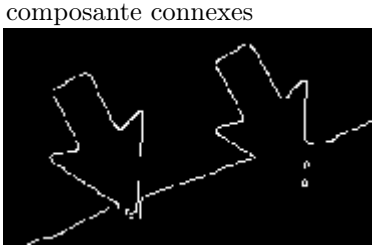
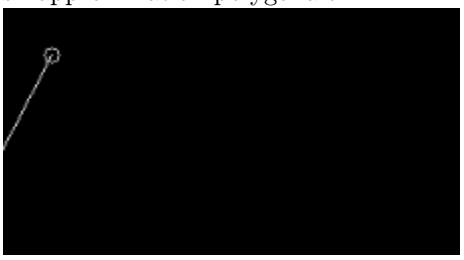


FIGURE 36 – Les deux pointeurs disparaissent après l'approximation polygonale



Nous remarquons également que parfois l'approximation polygonale déforme tellement une composante connexe que le résultat correspond à l'approximation polygonale d'un pointeur.

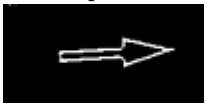
FIGURE 37 – Une composante connexe ne correspondant pas à un pointeur

FIGURE 38 – L'approximation polygonale de la composante connexe ci-dessus (*Figure 37*)

Nous remarquons également que parfois certains pointeurs sont répertoriés deux fois. Nous pensons que c'est à cause de la frontière qui sépare notre pointeur et l'arrière plan.

La frontière est généralement d'une couleur différente, nous extrayons donc deux fois la même composante connexes. Par exemple dans la figure ci-dessous (*Figure 39*), le pointeur a été répertorié deux fois lors de l'extraction des composantes connexes.

FIGURE 39 – Un pointeur noir avec une frontière blanche sur un arrière-plan noir



7 Conclusion et perspectives

7.1 Conclusion

Tout d'abord, nous tenons à remercier notre encadrant, M.Wendling qui nous a guidé et conseillé tout au long du projet. Ce projet a été pour nous une occasion d'en apprendre plus sur la vision informatique et la reconnaissance de forme.

Nous avons pu appliquer et tester des méthodes vues en cours. Nous avons ainsi pu mieux assimiler certains concepts. Par exemple, nous comprenons mieux le fonctionnement et l'intérêt des filtres de convolution en traitement d'image.

Nous avons également découvert des techniques utilisés dans le traitement d'image. Par exemple, un filtre efficace pour obtenir les contours d'une image est le filtre de Canny.

Nous nous sommes familiarisé avec différents outils utilisés pour la reconnaissance de forme. Par exemple nous sommes maintenant plus à l'aise avec la bibliothèque *OpenCV* en Python pour le traitement d'image et la bibliothèque *Scikit-learn* pour la classification.

Nous comprenons également mieux les limites de la classification non supervisée. Par exemple, il est assez compliqué d'évaluer l'efficacité d'une classification non supervisée, nous pouvons calculer l'inertie interclasse ou intraclasse mais cela n'est pas toujours pertinent.

7.2 Perspectives

Suite à ce projet, nous avons réalisé qu'il est possible de créer des modèles de classification non supervisé pour la détection de pointeurs. Cependant nous pensons qu'il est possible d'améliorer notre approche.

Tout d'abord, nous devons trouver un moyen de supprimer les éléments qui apparaissent plusieurs fois lors l'extraction des composantes connexes.

Ensuite, nous devons trouver une méthode qui permet de "protéger" nos pointeurs des éléments qui sont en contact avec pour éviter qu'ils soient inclues comme des éléments d'une composante connexe plus grande.

Une fois que nous avons réglé ces deux problèmes, nous pouvons utiliser plus de critères géométriques pour filtrer les composantes connexes.

Par exemple, nous pouvons chercher l'axe de symétrie de la flèche et calculer le degré de symétrie entre les deux cotés de l'axe.

Nous pouvons également essayer la classification en utilisant un descripteur différent du descripteur de Fourier générique ou alors en utilisant un autre modèle que celui des K-moyennes.

8 Bibliographie

1. Santosh, Kc & Wendling, Laurent & Antani, Sameer & Thoma, George. (2015). **Overlaid arrow detection for labeling biomedical image regions**. Intelligent Systems, IEEE. 31. 10.1109/MIS.2016.24.
2. L. Wendling and S. Tabbone, **"A new way to detect arrows in line drawings,"** in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 26, no. 7, pp. 935-941, July 2004, doi : 10.1109/TPAMI.2004.20.
3. Dengsheng Zhang, Guojun Lu, **Shape-based image retrieval using generic Fourier descriptor**, Signal Processing : Image Communication, Volume 17, Issue 10, 2002, Pages 825-848, ISSN 0923-5965.
4. Salvatore Tabbone, Thi Oanh Nguyen, Gérald Masini. **Une méthode de binarisation hiérarchique floue**. Colloque International Francophone sur l'Écrit et le Document - CIFED 2006, Sep 2006, Fribourg,Suisse. pp.43-48.
5. Satoshi Suzuki, Keiichi Abe, **Topological structural analysis of digitized binary images by border following**, Computer Vision, Graphics, and Image Processing, Volume 30, Issue 1, 1985, Pages 32-46, ISSN 0734-189X.