Traitement d'images
Compte-rendu des travaux pratiques

Clara Gros

Janvier 2020

Contents

1 Introduction : objectif de ce TP **2**

2 Génération d'images binaires **2**
2.1 Principe de la méthode d'Otsu 2
2.2 Application à une image 3

3 Caractérisation des images binaires **5**
3.1 Description de formes 5
3.2 Description des contours 7
3.2.1 Principe des descripteurs de Fourier 7
3.2.2 Influence du paramètre c_{max} 8

4 Classification **11**
4.1 Analyse en composante principale 11
4.1.1 Principe de la PCA 11
4.2 Détail de l'algorithme de PCA 12
4.2.1 Application au jeu de données 13
4.3 Méthode des K-means 14
4.4 Méthode des K-nearest-neighbors 17
4.5 Application à un nouveau set de données 17

5 Évaluation des performances **18**

6 Généralisation **20**

7 Conclusion **22**

1 Introduction : objectif de ce TP

Ces séances de travaux pratiques constituent une initiation à la classification d'image. L'objectif de ces travaux pratiques est donc de tester des méthodes de caractérisation de formes pour ensuite pouvoir classer les images selon plusieurs labels (plusieurs formes).

Plus en détail, à partir d'une base d'images que l'on veut classifier, il faut d'abord réaliser un pré-traitement pour générer des images binaires. À partir de ces images binaires, on calcule pour chaque images les descripteurs de Fourier. De ces descripteurs on peut ensuite extraire le contour de chaque image. Pour avoir une idée de la répartition des images dans l'espaces des labels, au préalable du traitement, on réalise une analyse en composante principale (PCA) pour trouver une "meilleure base" de représentation des données obtenue par combinaison linéaire de la base originale ce qui permet de faire de la réduction de dimensionalité et donc d'éliminer l'information redondante. Une fois ce pré-traitement effectué, on applique les algorithmes de Machine Learning classiques : les k-moyennes et les k plus proche voisins . Une validation des performances nous permet ensuite de comparer les méthodes. Enfin, pour aller plus loin, on propose en guise d'ouverture de tester notre algorithme sur une autre base d'images avec d'autres labels.

2 Génération d'images binaires

Avant de traiter les données, il faut convertir chaque image composée de nuances de gris en une image binaire noir et blanc. Pour cela, nous faisons appel à la fonction *imbinarize* (`https://fr.mathworks.com/help/images/ref/imbinarize.html?s_tid=doc_ta`) de $Matlab$.

La fonction *imbinarize(I)* créé une image binaire à partir d'une image grisée en remplaçant toutes les valeurs au dessus d'un certain seuil par des 1 et en affectant des 0 aux autres valeurs. Par défaut, *imbinarize* fixe le seuil en utilisant la méthode d'Otsu.

2.1 Principe de la méthode d'Otsu

La méthode d'Otsu est utilisée pour effectuer un seuillage automatique à partir de la forme de l'histogramme de l'image, ou la réduction d'une image à niveaux de gris en une image binaire. L'algorithme suppose alors que l'image à binariser ne contient que deux classes de pixels, (le premier plan et l'arrière-plan) puis calcule le seuil optimal qui sépare ces deux classes afin que leur variance intra-classe soit minimale.
Dans la méthode d'Otsu, le seuil qui minimise la variance intra-classe est recherché à partir de tous les seuillages possibles :

$$\sigma_w^2(t) = \omega_1(t)\sigma_1^2(t) + \omega_2(t)\sigma_2^2(t) \tag{1}$$

Les poids ω_i représentent la probabilité d'être dans la ième classe, chacune étant séparée par un seuil t. Finalement, les σ_i^2 σ_i^2 sont les variances de ces classes. Otsu

montre que minimiser la variance intra-classe revient à maximiser la variance inter-classe:

$$\sigma_b^2(t) = \sigma^2 - \sigma_w^2(t) = \omega_1(t)\omega_2(t)\left[\mu_1(t) - \mu_2(t)\right]^2 \tag{2}$$

qui est exprimée en termes des probabilités de classe ω_i ω_i et des moyennes de classes μ_i μ_i qui à leur tour peuvent être mises à jour itérativement. Cette idée conduit à un algorithme efficace.

Algorithm 1: Seuillage (méthode d'Otsu)

Result: Seuil t
initialisation;
Calculer l'histogramme et les probabilités de chaque niveau d'intensité;
Définir les $\omega_i(0)$ $\omega_i(0)$ et $\mu_i(0)$ $\mu_i(0)$ initiaux;
Initialisation du seuil : $t = 1$;
Initialisation de la variance $\sigma_b^2(t) = 0$;
while $t <$ *intensité max* **do**
    Mettre à jour ω_i et μ_i;
    Calculer $\sigma_b^2(t)$;
end
Le seuil désiré t correspond au $\sigma_b^2(t)$ maximum.

2.2 Application à une image

Le graphique ci dessous représente l'histogramme de l'image non binaire. En appliquant le filtre d'Otsu, on arrive à déterminer un seuil optimal représenté ci dessous pour pouvoir affecter une valeur binaire à chaque pixel de l'image.

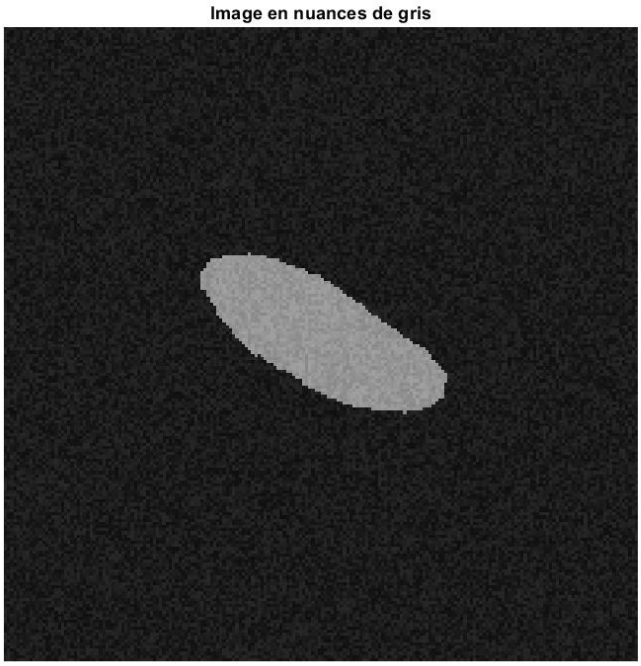
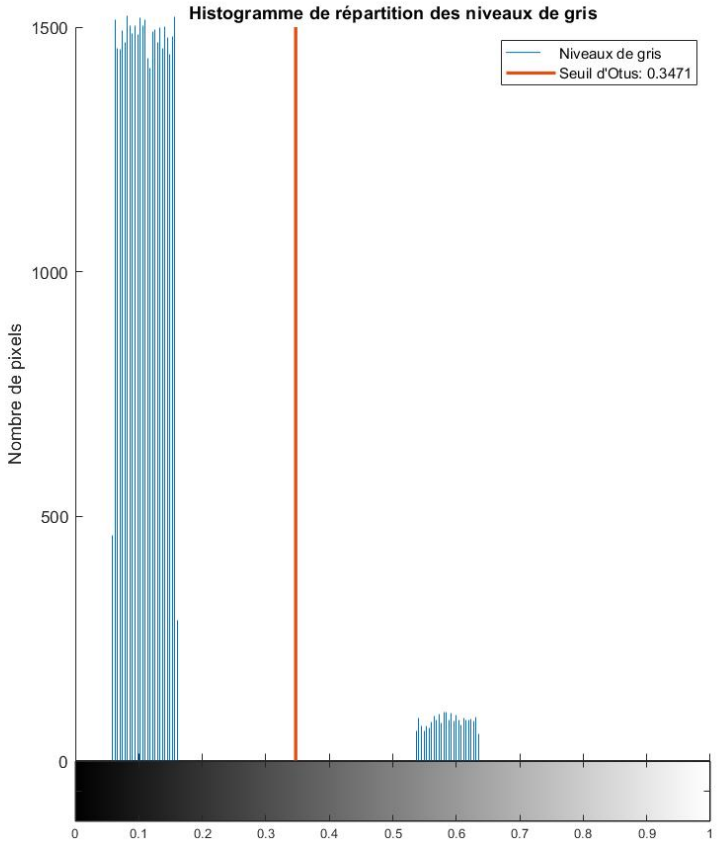
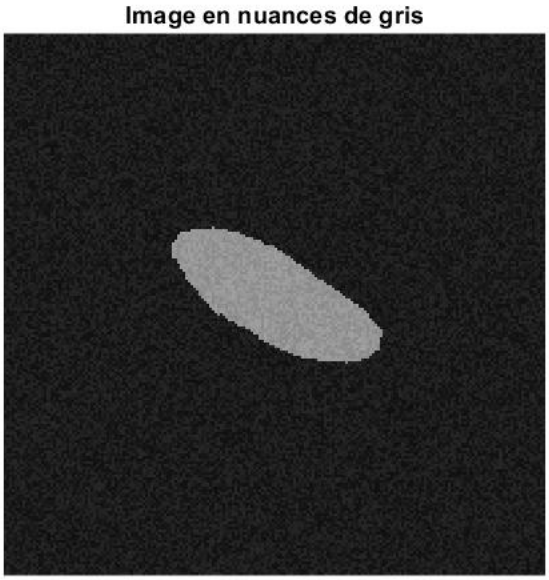
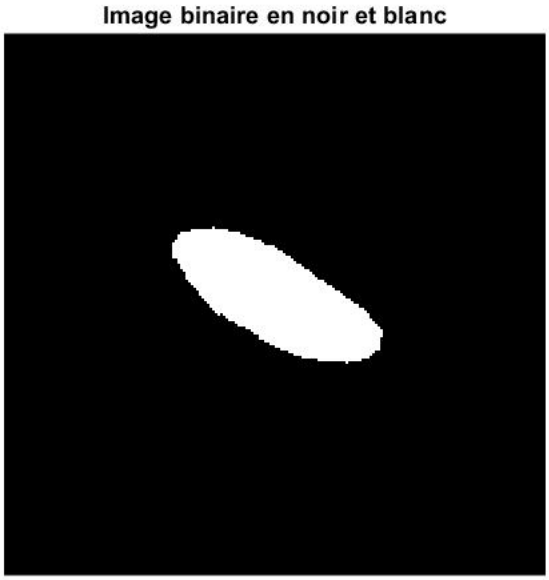


Image en nuances de gris


Histogramme de répartition des niveaux de gris
Niveaux de gris
Seuil d'Otus: 0.3471
Nombre de pixels
1500
1000
500
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1





3 Caractérisation des images binaires

3.1 Description de formes

Avant de nous intéresser aux descripteurs de Fourier et d'expliquer leur utilité, il est d'abord possible de différentier des formes en fonction d'autres paramètres géométriques. En utilisant la fonction *regionprops* sur une image binaire, il est possible d'obtenir des paramètres indépendants de la translation, la rotation et l'homothétie sur les formes sans avoir à extraire un contour `https://fr.mathworks.com/help/images/ref/regionprops.html?s_tid=doc_ta`.

`https://perso.telecom-paristech.fr/bloch/P6Image/FT_Formes.pdf`

Nous observons bien que pour les objets que nous avons à traiter, comme ce sont de petits objets semblables, ces paramètres ne sont pas forcément pertinents pour pouvoir faire une bonne classification. On se propose donc dans la suite de caractériser la

forme par son contour notamment en s'appuyant sur les descripteurs de Fourier pour une meilleure classification.

Nous allons trier selon les critères suivants qui ne dépendent ni de la translation, ni de la rotation et ni de l'homothétie :

- Circularité : spécifie la rondeur des objets. La valeur de circularité est calculée comme $4 \cdot Area \cdot \pi / Perimeter^2$. Pour un cercle parfait, la valeur de circularité est 1.
- Excentricité : rapport de la distance entre les foyers de l'ellipse et la longueur de son axe principal. La valeur est comprise entre 0 et 1. (0 et 1 sont des cas dégénérés. Une ellipse dont l'excentricité est 0 est en fait un cercle, tandis qu'une ellipse dont l'excentricité est 1 est un segment de ligne.)
- Rapport du plus petit et plus grand axe
- Extent : rapport entre les pixels de la forme et les pixels de la bounding box. Calculé comme l'aire de la forme divisée par l'aire de la bounding box. La bounding box correspond au plus petit rectangle contenant la la forme.
- Solidity :rapport entre les pixels de la forme et les pixels de la convex area. La convex area correspond au plus petit polygone convexe pouvant contenir la forme.

Le tableau ci-dessous résume les résultats obtenus pour une image de chaque classe (le nom de la classe correspondant à la forme de l'image).

Labels	Circularité	Excentricité	Petit axe/Grand axe	Extent	Solidity
Ovale	0.6898	0.9294	0.3690	0.5354	0.9647
H	0.2157	0.3330	0.9429	0.5122	0.5870
Rond	0.9532	0.1832	0.9831	0.7718	0.9811
Rectangle	0.7395	0.7813	0.6241	0.7710	0.9506
Étoile	0.2355	0.3616	0.9324	0.3845	0.4189
Y	0.3818	0.5175	0.8557	0.4162	0.6490

Grâce à toutes ses caractérisations de formes, on s'aperçoit que les méthodes de classification de clustering sont quasiment obsolètes dans la mesure où il est très facile de distinguer un disque des autres formes en regardant le critère de circularité qui est très supérieur aux autres classes. De même, l'excentricité permettrait facilement d'identifier un ovale dont l'excentricité est très proche de 1.

En combinant ces différents critères, on pourrait peut être réussir à classer les images selon leur formes. Cependant, des formes telles que le Y ou le H ont des caractéristiques semblables pour chaque critère. Trier des images provenant des deux classes ne serait donc pas possible en utilisant seulement des critères géométriques. C'est pourquoi par la suite on se base sur des méthodes de clustering de Machine Learning une fois le contour de l'image extrait.

3.2 Description des contours

Pour extraire le contour de la forme, on se propose d'utiliser la fonction *bwboundaries* (`https://fr.mathworks.com/help/images/ref/bwboundaries.html?s_tid=doc_ta`) de $Matlab$ qui prend en paramètre une image binaire. *bwboundaries(BW)* trace les contours extérieurs des objets et les trous dans ces objets. Dans notre cas, il ne s'agira que de contours. La fonction retourne une unique cellule dans notre cas ce qui correspond aux locations des pixels du contour de la forme.

On ne s'intéressera pas à l'algorithme de traçage de Moore-Neighbor modifié par le critère d'arrêt de Jacob qui est implémenté dans la fonction bwboundaries. Cette fonction se base sur les fonction de contour présentées dans la première édition de DIGITAL IMAGE PROCESSING USING MATLAB, BY GONZALEZ, R. C., R. E. WOODS, AND S. L. EDDINS, NEW JERSEY, PEARSON PRENTICE HALL, 2004.

Avec la fonction bwboundaries, on génère la suite de nombres complexes des "pixels contour". Par exemple un pixel contour de coordonnée (a, b) se verra attribuer le nombre complexe $a + j * b$ qui permettra le calcul des descripteurs de Fourier (fonction $dfdir$ fournié par l'énoncé). A partir de ces descripteurs, nous reconstruirons le contour de l'image grâce à la fonction $dfinv$ fournie par l'énoncé.

3.2.1 Principe des descripteurs de Fourier

Granlund a introduit les descripteurs de Fourier en utilisant la représentation complexe en 1972. Cette méthode garantit qu'une courbe fermée correspondra à n'importe quel ensemble de descripteurs. La forme est maintenant décrite par un ensemble de N sommets complexes $z(i) : i = 1, \ldots, N$ correspondant à N points du contour. Les descripteurs de Fourier $c(k) : k = -N/2 + 1, \ldots, N/2$ sont les coefficients de la transformée de Fourier de z.

$$z_i = \sum_{k=-N/2+1}^{N/2} c_k \exp(2\pi j \frac{ki}{N}) \tag{3}$$

La relation inverse existe entre $c(k)$ et $z(i)$:

$$c_k = \frac{1}{N} \sum_{i=1}^{N} c_i \exp(-2\pi j \frac{ik}{N}) \tag{4}$$

La plage de k peut être limitée à $\frac{-N}{2} + 1, \frac{N}{2}$: selon le théorème de Shannon, la fréquence la plus élevée est obtenue pour $k = \frac{N}{2}$, et tout $c(k)$ avec $k > \frac{N}{2}$ serait redondant puisque nous utilisons une représentation discrète du contour. Les descripteurs $c(k)$ décrivent le contenu fréquentiel de la courbe: une valeur de k proche de zéro décrira des informations basse fréquence, une forme approximative, et les fréquences plus élevées décriront des détails. Pour $k = 0$, $c(k)$ représente la position du centre de gravité de la forme. Ce terme n'est pas intéressant pour la description de la forme. La première composante de fréquence, $c(k)$ pour $k = 1$, décrit la taille de la forme. Si tous les autres composants sont définis sur zéro, la forme devient un cercle. On

peut utiliser ce composant pour normaliser l'ensemble des descripteurs de Fourier, afin qu'ils restent constants après une homothétie sur la forme. Les autres composantes de fréquence effectueront des modifications sur le cercle décrit par $c(1)$.

Les descripteurs de Fourier complexes permettent de choisir la précision obtenue dans la description d'une courbe. Si nous ne conservons qu'un sous-ensemble de descripteurs de basses fréquences, nous obtenons une courbe qui se rapproche simplement du contour d'une forme. En augmentant le nombre de composants dans la description, les hautes fréquences sont également rendues et des courbes ou des détails nets peuvent être générés.

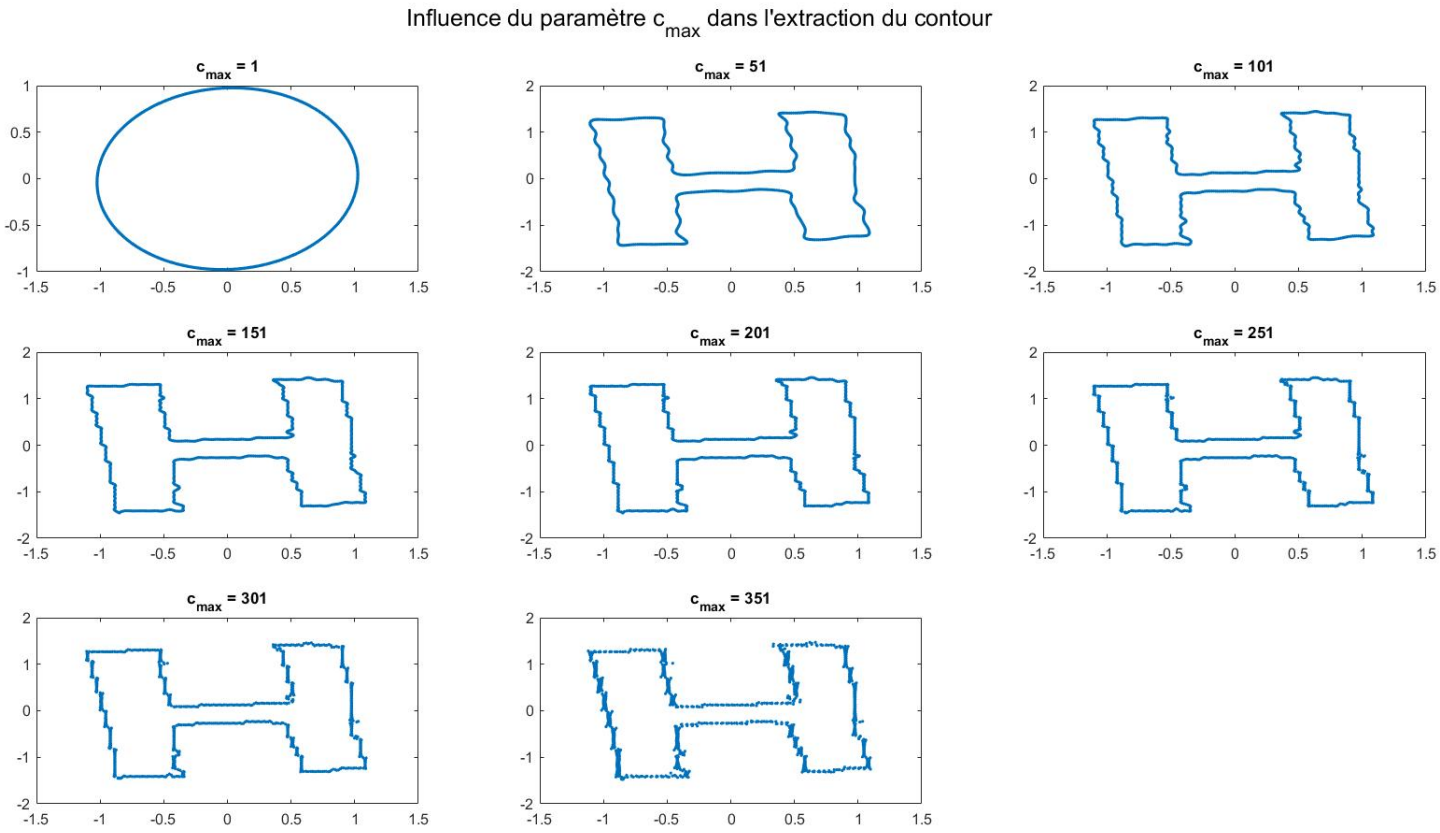
Propriétés de la méthode d'extraction de contour avec les descripteurs de Fourier

- Invariants par translation
- Invariants par rotation
- Invariants par changement d'origine

Nous pouvons donc conclure en justifiant l'utilisation de cette méthode pour comparer et classifier des formes car dans notre cas, comparer des formes se réduit à comparer des descripteurs par ordre croissant. Il ne nous reste plus qu'à fixer la précision de contour obtenu souhaitée en réglant le nombre de coefficients.

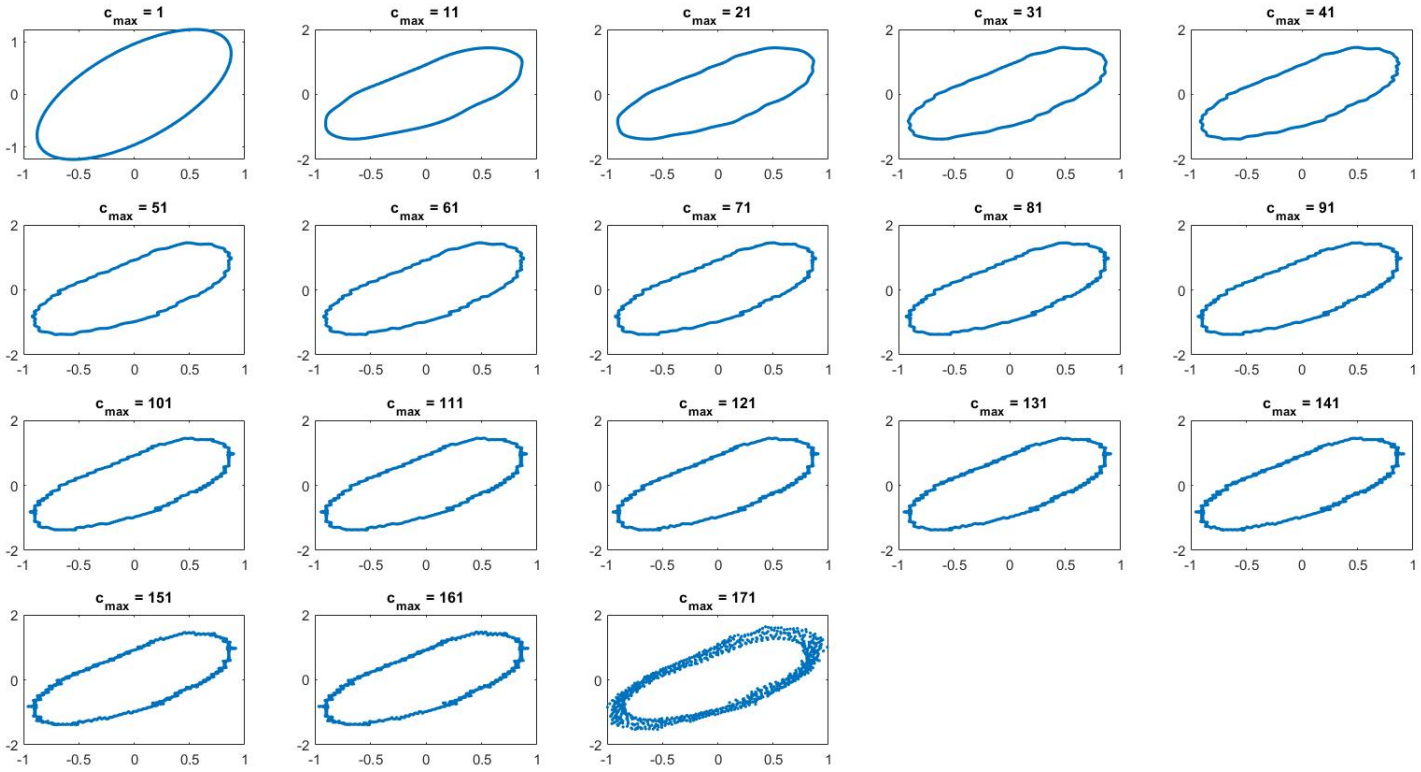
3.2.2 Influence du paramètre c_{max}

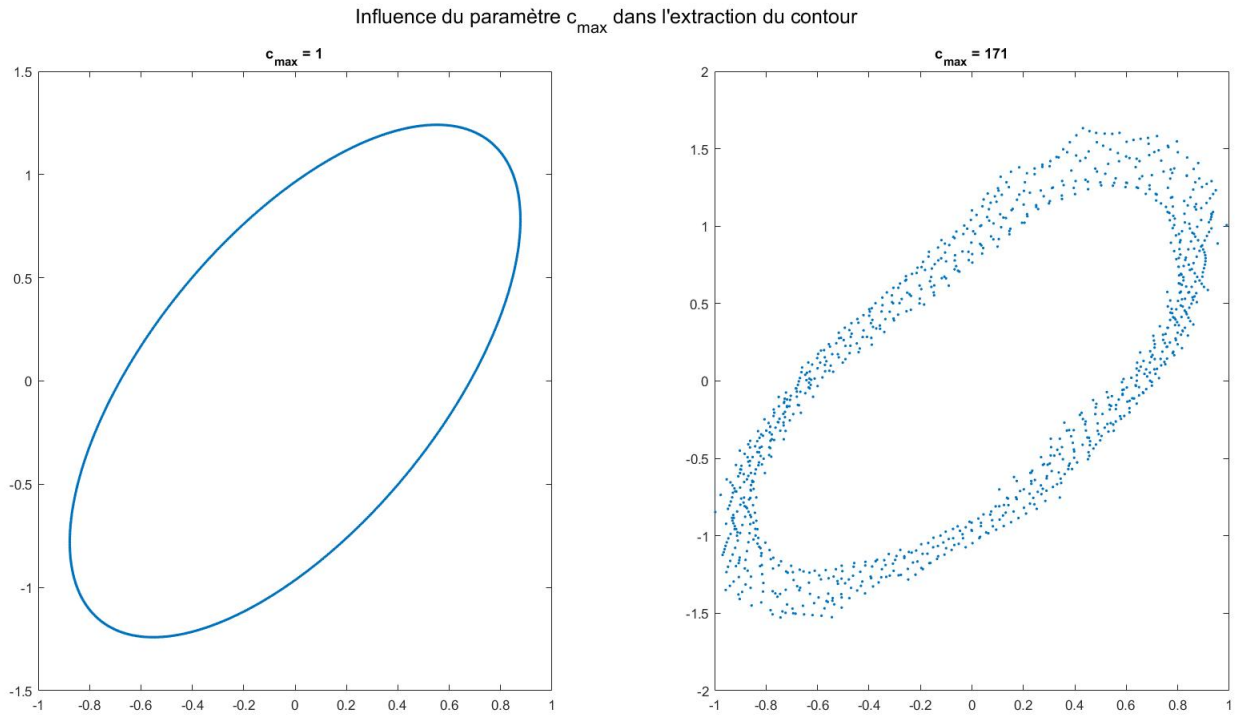
Dans $Matlab$, j'ai mis au point un algorithme pour tester l'influence du nombre de coefficients dans la fonction *dfdir* qui génère les descripteurs de Fourier. Ci-dessous plusieurs graphiques qui rendent compte de l'influence de ce paramètre.



En analysant les images, on remarque que plus le nombre de coefficients est élevé, plus la forme est complexe. En effet, on voit bien que pour $c_max = 1$, le contour obtenu pour un H a la même forme que le contour obtenu pour un réel ovale. Il faut donc au moins $c_max > 1$ pour pouvoir distinguer les images entre elles. De plus, les coefficients d'ordre élevés vont être à l'origine des détails fins sur la courbe. Ainsi, plus on augmente c_{max} i.e. le nombre de coefficients, plus la reconstruction du contour est sensible aux variations du vraie contour. Pour les images telles que le H, cela ne pose visiblement pas de problème majeur cependant pour la classe d'images ovales, un c_{max} élevé brouille la limite du contour comme en témoigne les deux graphiques ci-dessous. Pour poursuivre l'analyse, on va donc se limiter à un petit c_{max} qui est assez important pour pouvoir faire la différence entre toutes les figures mais qui fonctionne pour toutes les classes d'images dont nous disposons. Dans le code $Matlab$, on prendra $c_{max} =$

Influence du paramètre c_{max} dans l'extraction du contour





4 Classification

4.1 Analyse en composante principale

4.1.1 Principe de la PCA

L'analyse en composantes principales (ACP), ou principal component analysis (PCA) en anglais, permet d'analyser et de visualiser un jeu de données contenant ici des images décrits par plusieurs variables quantitatives (descripteurs de Fourier).

C'est une méthode statistique qui permet d'explorer des données dites multivariées. Chaque variable pourrait être considérée comme une dimension différente. S'il y a plus de 3 variables dans le jeu de données, il pourrait être très difficile de visualiser les données dans une "hyper-espace" multidimensionnelle. L'analyse en composantes principales est utilisée pour extraire et de visualiser les informations importantes contenues dans une table de données multivariées. L'ACP synthétise cette information en seulement quelques nouvelles variables appelées composantes principales. Ces nouvelles variables correspondent à une combinaison linéaire des variables originels. Le nombre de composantes principales est inférieur ou égal au nombre de variables d'origine.

L'information contenue dans un jeu de données correspond à la variance ou l'inertie totale qu'il contient. L'objectif de l'ACP est d'identifier les directions (i.e., axes principaux ou composantes principales) le long desquelles la variation des données est maximale. En d'autres termes, l'ACP réduit les dimensions d'une donnée multivariée à deux ou trois composantes principales, qui peuvent être visualisées graphiquement, en perdant le moins possible d'information.

4.2 Détail de l'algorithme de PCA

- **Standardisation** : Le but de cette étape est de standardiser la gamme des variables initiales continues afin que chacune d'entre elles contribue également à l'analyse. Mathématiquement, cela peut être fait en soustrayant la moyenne et en divisant par l'écart-type pour chaque valeur de chaque variable.

$$z = \frac{value - mean}{standardDeviation} \tag{5}$$

  Une fois la normalisation effectuée, toutes les variables seront transformées à la même échelle.

- **Calcul de la matrice de covariance** : Le but de cette étape est de comprendre comment les variables de l'ensemble de données d'entrée diffèrent de la moyenne les unes par rapport aux autres, ou en d'autres termes, de voir s'il existe une relation entre elles. Parce que parfois, les variables sont fortement corrélées de manière à contenir des informations redondantes. Ainsi, afin d'identifier ces corrélations, nous calculons la matrice de covariance.

- **Calcul des vecteurs et valeurs propres de la matrice de covariance** : Géométriquement parlant, les composantes principales représentent les directions des données qui expliquent une variance maximale, c'est-à-dire les lignes qui captent la plupart des informations des données. La relation entre la variance et l'information ici est que, plus la variance portée par une ligne est grande, plus la dispersion des points de données le long de celle-ci est grande, et plus la dispersion le long d'une ligne est grande, plus elle dispose d'informations. Mathématiquement parlant, ce sont les vecteurs propres et les valeurs propres qui sont derrière tout cela, car les vecteurs propres de la matrice de Covariance sont en fait les directions des axes où il y a le plus de variance (le plus d'informations) et que nous appelons Composantes Principales. Et les valeurs propres sont simplement les coefficients attachés aux vecteurs propres, qui donnent la quantité de variance portée dans chaque composante principale. En classant vos vecteurs propres par ordre de leurs valeurs propres, du plus haut au plus bas, vous obtenez les principaux composants par ordre d'importance.

Finalement nous obtenons une matrice de changement de base où chaque vecteur correspond à un axe principal, sur laquelle nous allons pouvoir projeter notre jeu de données initial.

4.2.1 Application au jeu de données

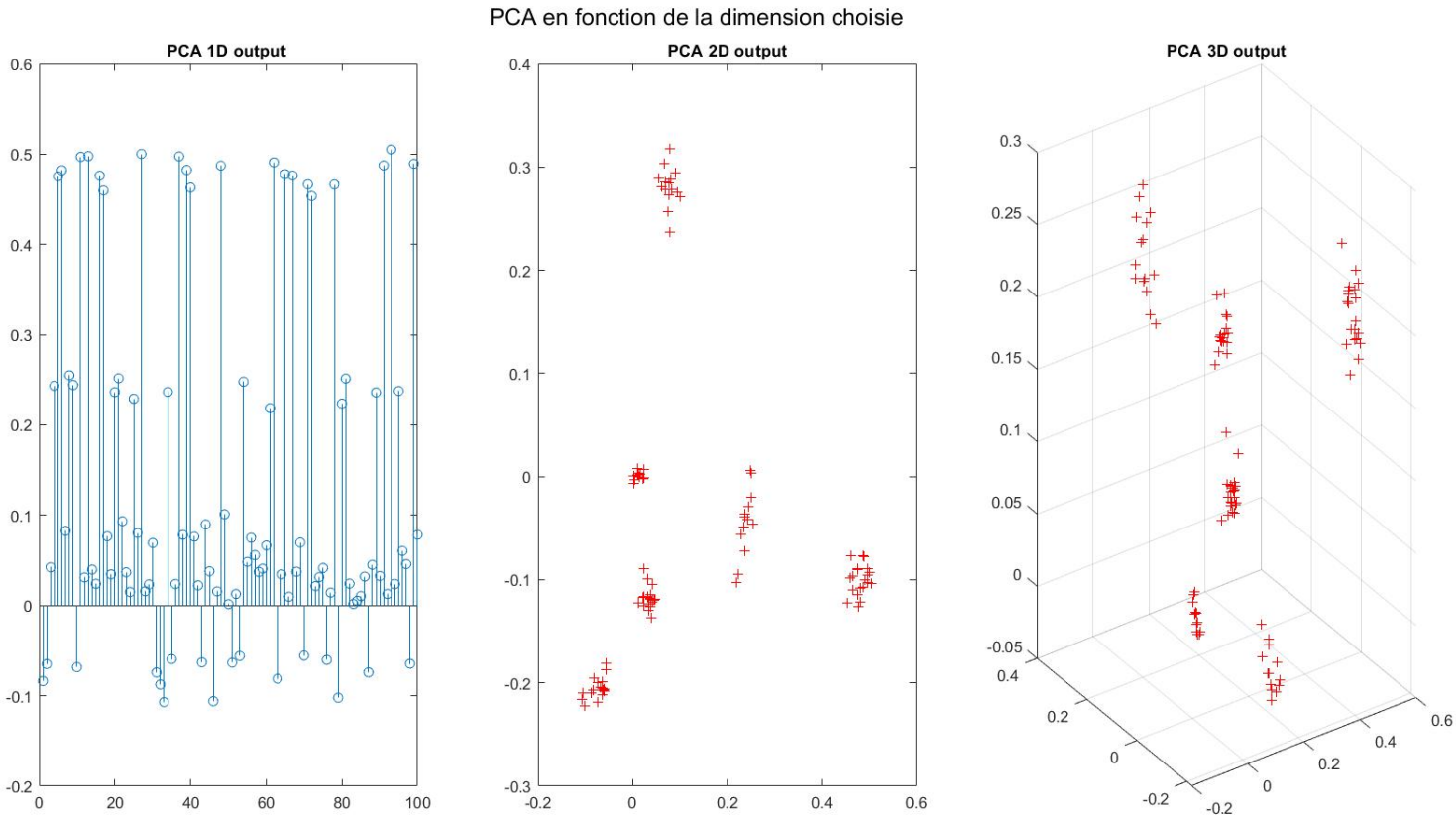
En pratique, dans $Matlab$, on applique la fonction pca. Pour cela, dans une boucle, on récupère une liste des valeurs absolue des descripteurs de Fourier pour obtenir des attributs indépendants de la rotation.pca renvoie les coefficients de composante principale pour la matrice de données entrée en paramètre. La matrice des coefficients est carrée. Chaque colonne contient des coefficients pour un composant principal, et les colonnes sont dans l'ordre décroissant de variance des composants. Par défaut, pca centre les données et utilise l'algorithme de décomposition en valeurs singulières (SVD).

Voici ensuite les étapes que j'ai réalisé dans le programme $Matlab$:

- **Re conditionner les données le long des axes principaux composants** : Le but est d'utiliser la matrice de changement de base pour réorienter les données des axes d'origine vers celles représentées par les composantes principales (d'où le nom Analyse en composantes principales). Cela peut être fait en multipliant la transposition de l'ensemble de données d'origine par la transposition du vecteur caractéristique.

$$FinalDataSet = pca^{-1} \cdot StandardizedOrginalDataSet^t \tag{6}$$

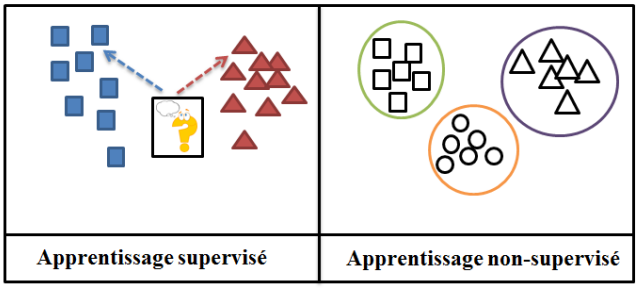
- **Sélection des axes principaux - Réduction de la dimensionnalité** : dans cette étape, ce que nous faisons est de choisir de rejeter les composants qui ont le moins d'importance (de faibles valeurs propres) et de former avec les autres le vecteur de données sur lequel on va appliquer les algorithmes de Machine Learning.
  Dans $Matlab$, j'ai choisi dans un premier temps de sélectionner les 3 premiers composants principaux.



Voici la répartition de nos données dans la nouvelle base en fonction du nombre d'axes principaux que j'ai souhaité conservé. A partir de la dimension 2, on observe une répartition des données beaucoup plus organisée en différents clusters, on comprend alors l'intérêt d'appliquer l'algorithme de pca avant d'utiliser une méthode de classification basée sur le clustering. Cependant constate qu'il n'y a pas une grande différence entre une dimension 2 et une dimension 3. Dans une section Validation des performances, nous testerons l'influence de l'ajout ou du retrait d'une dimension.

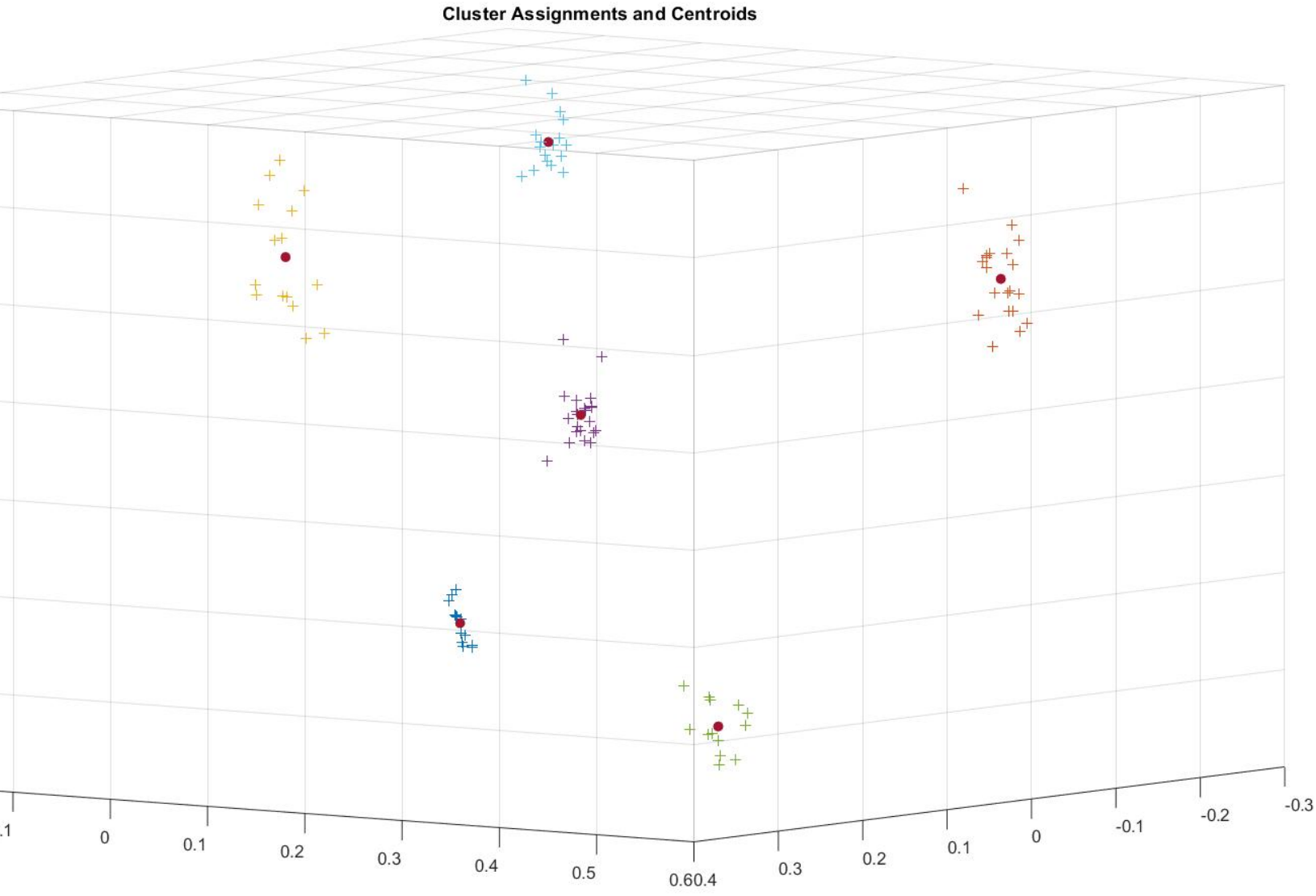
4.3 Méthode des K-means

Méthode non supervisée La méthode des $kmeans$ est une méthode de clustering très utilisée et relève de l'apprentissage non supervisé. L'apprentissage non supervisé consiste, pour $Matlab$, à trouver des structures sous-jacentes à partir de données non étiquetées. Puisque les données ne sont pas étiquetées, il n'est pas possible d'affecter au résultat de l'algorithme utilisé un score d'adéquation. Cette absence d'étiquetage (ou d'annotation) est ce qui distingue les tâches d'apprentissage non-supervisé des tâches d'apprentissage supervisé.



Principe de l'algorithme kmeans La fonction *kmeans* `https://fr.mathworks.com/help/stats/k-means-clustering.html?searchHighlight=kmeans&s_tid=doc_srchtitle` partitionne les données en k clusters s'excluant mutuellement et renvoie l'index du cluster auquel elle affecte chaque observation. *kmeans* traite chaque observation dans les données comme un objet qui a une position dans l'espace. La fonction trouve une partition dans laquelle les objets de chaque cluster sont aussi proches les uns des autres que possible et aussi éloignés des objets des autres clusters que possible. Donc, dans chaque cluster, *kmeans* minimise la somme des distances entre le centre de gravité et tous les objets membres du cluster. Par défaut, *kmeans* utilise l'algorithme k-means ++ `https://fr.mathworks.com/help/stats/kmeans.html#bueftl4-1` pour initialiser les centroïdes de cluster et la distance euclidienne pour déterminer les distances.

Résultat de l'algorithme Pour choisir le nombre de clusters désirés, j'applique la fonction *evacluster* de *Matlab* qui évalue le nombre optimal de cluster à l'aide du critère d'évaluation des clusters de Calinski-Harabasz [1] CALINSKI, T., AND J. HARABASZ. "A DENDRITE METHOD FOR CLUSTER ANALYSIS." COMMUNICATIONS IN STATISTICS. VOL. 3, NO. 1, 1974, PP. 1–27. La fonction suggère alors d'exécuter le programme en séparant les données en 6 clusters distincts. Ci dessous figure le résultat graphique de l'algorithme *kmeans*. On peut observer la position des centroïdes ainsi que la répartition des données dans chaque cluster ce qui semble à permière vue être cohérent.



Néanmoins, puisque c'est une méthode d'apprentissage non supervisé, nous ne sommes pas encore capables d'apparier un cluster à un label (à une forme). C'est donc ce que nous allons faire dans la section suivante.

4.4 Méthode des K-nearest-neighbors

Principe de l'algorithme Par opposition à un algorithme d'apprentissage automatique non supervisé comme $kmeans$, la méthodes des k plus proches voisins est un algorithme d'apprentissage automatique supervisé. L'algorithme s'appuie sur des données d'entrée étiquetées pour apprendre une fonction qui produit une sortie appropriée lorsqu'il reçoit de nouvelles données non étiquetées. `https://fr.mathworks.com/help/stats/fitcknn.html?searchHighlight=fitcknn&s_tid=doc_srchtitle#buetfbv-6`

Pour pouvoir effectuer une prédiction, K-NN se base sur le jeu de données pour produire un résultat. Pour une observation qu'on souhaite prédire, qui ne fait donc pas parti du jeu de données, l'algorithme va chercher les K instances du jeu de données les plus proches de notre observation. Ensuite pour ces K voisins, l'algorithme se basera sur leurs variables de sortie y pour calculer la valeur de la variable y de l'observation que l'on souhaite prédire.

Par ailleurs si K-NN est utilisé pour la classification, c'est le mode des variables y des K plus proches observations qui servira pour la prédiction.

Le choix de la valeur K à utiliser pour effectuer une prédiction avec K-NN, varie en fonction du jeu de données. En règle générale, moins on utilisera de voisins (un nombre K petit) plus on sera sujette au sous apprentissage (underfitting). Par ailleurs, plus on utilise de voisins (un nombre K grand) plus, sera fiable dans notre prédiction. Toutefois, si on utilise K nombre de voisins avec K=N et N étant le nombre d'observations, on risque d'avoir du overfitting et par conséquent un modèle qui se généralise mal sur des observations qu'il n'a pas encore vu.

Algorithm 2: Algorithme des K plus proches voisins

Data: un ensemble de données D et un nombre entier K
input : Une nouvelle observation X
output: Classification de X
initialisation;
foreach *element* e *of the set* D **do** $computeDistance(e, X)$;
Retenir les K observations du jeu de données D les proches de X;
Prendre les labels l des K observations retenues;
Renvoie le mode des K labels;

4.5 Application à un nouveau set de données

Dans cette section, on se propose de mettre en place l'algorithme supervisé de Machine Learning des k nearest neighbors. Tout d'abord dans Matlab, on génère un deuxième set de données à partir des mêmes images de références et l'objectif est de classer ces images en fonction de leur label (forme). Nous allons faire cela pour deux jeu de données d'apprentissage dans le but de tester la performance de chaque méthode.

1. Méthode exploitant le résultat des k-moyennes Comme le titre l'indique, la première méthode consiste à choisir les centroïdes obtenus par la méthodes des $kmeans$

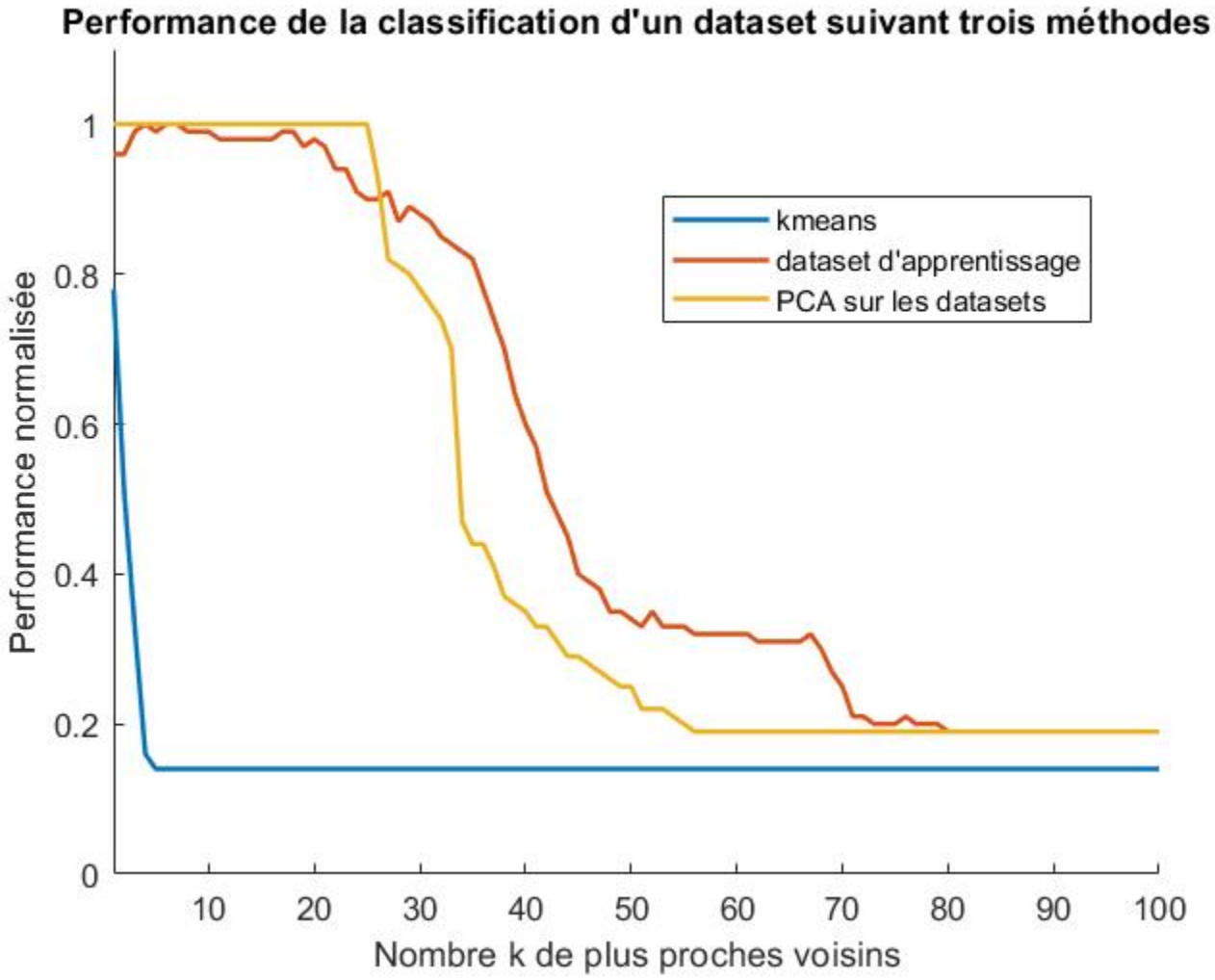
comme jeu de données d'apprentissage. En pratique, une fois que les centroîdes ont été calculé par l'algorithme des k-moyennes, on appelle la fonction $knnsearch$ de $Matlab$ qui trouve les k plus proches voisins d'un dataset (ici les centroïdes) dans un autre dataset (ici le dataset des données `https://fr.mathworks.com/help/stats/knnsearch.html?s_tid=doc_ta`. Ainsi, on peut prendre comme données d'apprentissage les images dont les coordonnées dans le plan des principaux composants sont quasi semblables à celles des centroïdes. Une fois ces données étiquetées, il suffit simplement de créer l'objet $ClassificationKNN$ pour pouvoir prédire les labels des images test.

2. Méthode exploitant directement les données de la base d'apprentissage La deuxième méthode de classification consiste à choisir comme jeu de données d'apprentissage le dataset de la base d'apprentissage tout entier. Il suffit ensuite de créer un deuxième objet $ClassificationKNN$ pour prédire les labels du dataset test. La différence entre les deux méthodes réside donc dans la différence du nombre d'images qui consitituent la base d'apprentissage (6 vs 100).

3. Méthode exploitant les jeu de données pré traités (pca) J'ai voulu ajouter une troisième méthode de classifictaion qui consiste simplement à reprendre les jeux de données de la deuxième méthode et à les pré traiter en leur appliquant l'algorithme de pca. Il sera donc intéressant dans la section suivante de comparer et commenter les résultats de ces trois méthodes pour plusieurs valeurs du paramètre k qui représente le nombre de plus proches voisins utilisés dans l'algorithme $fitcknn$.

5 Évaluation des performances

Pour évaluer les performances des 3 méthodes à k fixé, il suffit de regarder après la prédiction pour chaque image si le label prédit est égal au label fournit lors de la génération du dataset d'observation. De là en déduit un pourcentage de performance. Ci-dessous se trouve donc les performances pour chaque méthodes et pour plusieurs valeurs du paramètre k allant de 1 à 5.



Pour toutes les méthodes, le graphique confirme le fait qu'il ne faut pas avoir un paramètre k trop grand ie trop proche du nombre de données dans le dataset d'apprentissage car plus k est grand, plus le nombre de voisins pris en considération pour classer une image va être proche du dataset tout entier ce qui va tendre vers une prédicition équiprobable entre tous les labels (ici 0.1667).

Tout d'abord, pour des k raisonnables, concernant la première méthode exploitant les résultats de $kmeans$ il est normal de constater que plus k augmente plus les performances de l'algorithme diminuent car en ne sélectionnant que les centroïdes, il n'est pas utile de regarder plus d'un voisin : on cherche à faire de la quantification vectorielle avec donc $k = 1$.

L'algorithme des plus proches voisins est strictement plus performant en terme d'exactitude de la prédiction que la précédente méthode (même pour $k = 1$).

Le troisième algorithme où les deux datasets sont d'abord transformées avec la pca est beaucoup plus efficace car il semble que pour des k assez petits, la performance est toujours égale à 100%. Pour se persuader de l'efficacité de cet algorithme, il faudrait le tester sur plusieurs différents datasets pour s'assurer qu'il n'y pas d'overfitting. Ca sera l'objectif de la prochaine section dans laquelle on se propose de générer un dataset à partir d'autres images de référence.

Un des critères également important en Machine Learning est le temps d'exécution de la méthode de classification. A l'aide des fonction tic et toc de $Matlab$ il est possible de déterminer le temps d'exécution de chaque méthode. Le tableau ci-dessous montre ces données et révèle qu'en plus d'être moins performante, la première méthode est plus longue à implémenter. En revanche les deux autres méthodes sont à peu près équivalentes. Selon les applications, il faudra donc faire un compromis entre rapidité d'exécution et performance de la prédiction pour faire un choix quant à la méthode la mieux adaptée à la situation.

Durée d'exécution de la méthode de classification pour 100 valeurs de k(sec)

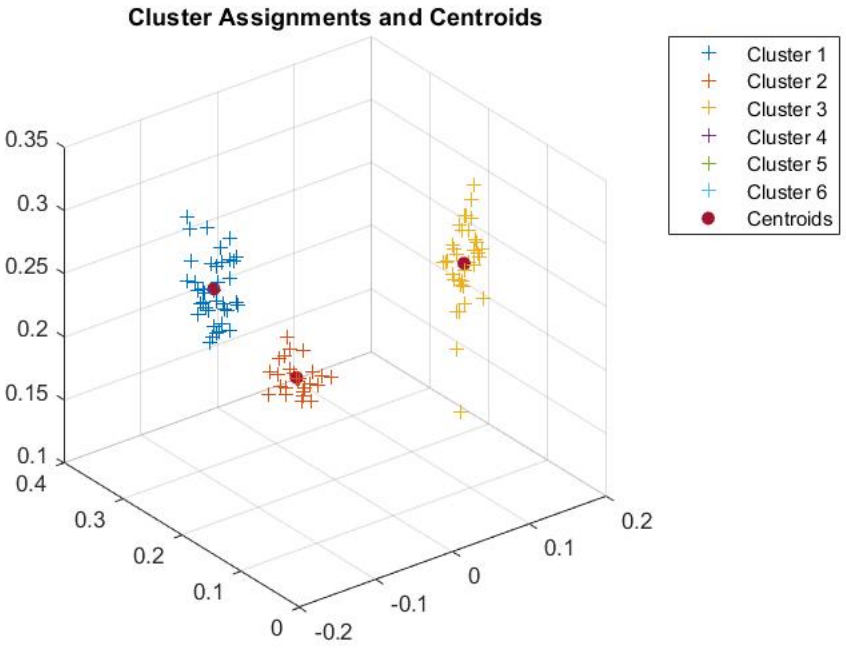
kmeans (1)	Dataset d'apprentissage (2)	PCA sur les datasets (3)
2.338897	1.410908	1.987712

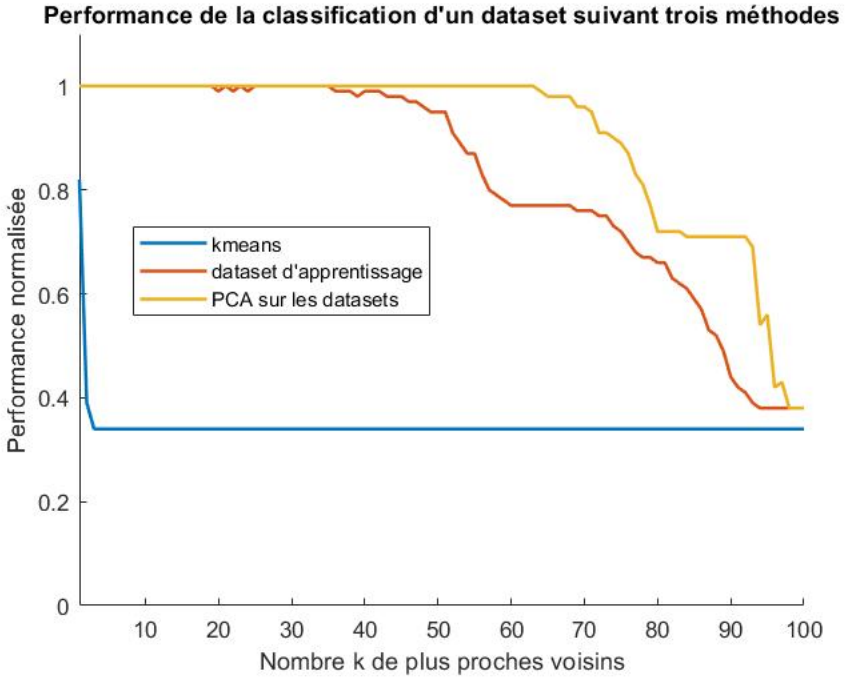
6 Généralisation

Figure 1: Images références du nouveau dataset

Pour généraliser mon propos, j'ai choisi 3 silhouette d'animaux que l'on va essayer de classifier avec les méthodes décrites précédemment.

Voici les résultats en termes de performances de prédiction et de temps de calcul pour chaque méthode.


Cluster Assignments and Centroids
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Cluster 6
Centroids
0.35
0.3
0.25
0.2
0.15
0.1
0.4
0.3
0.2
0.1
0
-0.2
-0.1
0
0.1
0.2


Performance de la classification d'un dataset suivant trois méthodes
Performance normalisée
Nombre k de plus proches voisins
kmeans
dataset d'apprentissage
PCA sur les datasets
1
0.8
0.6
0.4
0.2
0
10
20
30
40
50
60
70
80
90
100

Durée d'exécution des méthodes de classification pour 100 valeurs de k(sec)

kmeans (1)	Dataset d'apprentissage (2)	PCA sur les datasets (3)
3.472418	1.527891	2.441995

Dans cet exemple, pour une valeur de k faible, il semble que c'est la méthode 2 qui offre un meilleur rapport $\frac{performance}{tempsExécution}$.

7 Conclusion

En conclusion, après une préparation pour bien présenter les images sous forme binaire, on extrait des informations sur le contour de l'image grâce aux descripteurs de Fourier. A partir de ces variables définies pour chaque image, on peut soit décider d'appliquer la méthode de Machine Learning supervisée de $knearestneighbors$ ou choisir de pré-traiter les données par une analyse pca pour ensuite appliquer $knearestneighbors$.
La section précédente qui généralise notre étude à un autre set de données montre que ces algorithmes sont performants dans des contextes plus généraux. Cependant, j'ai été confronté à un problème lors de la sélection d'image : en effet, certaines images trop complexes n'ont qu'un nombre très faible de descripteurs de Fourier ce qui "fausse" l'appréciation des contours (c_{max} est limité) et cela empêche d'appliquer nos méthodes. Pour résoudre ce problème, on pourrait par exemple mieux pré traiter l'image pour soigner ses contours.