Rapport des TP de Traitement d'image.

Tariq Berrada Ifriqi

January 20, 2020

Contents

Introduction

Ce rapport à pour objectif de présenter le travail qui a été effectué lors des séances de travaux pratiques, notamment en expliquant les méthodes utilisés ainsi que les résultats obtenus et les interprétations possibles de ces résultats.

1 Génération et traitement de la base

On commence par génerer une base de 100 images à partir des formes élémentaires présents dans le dossier `appr` en utilisant le script `genere_base`.

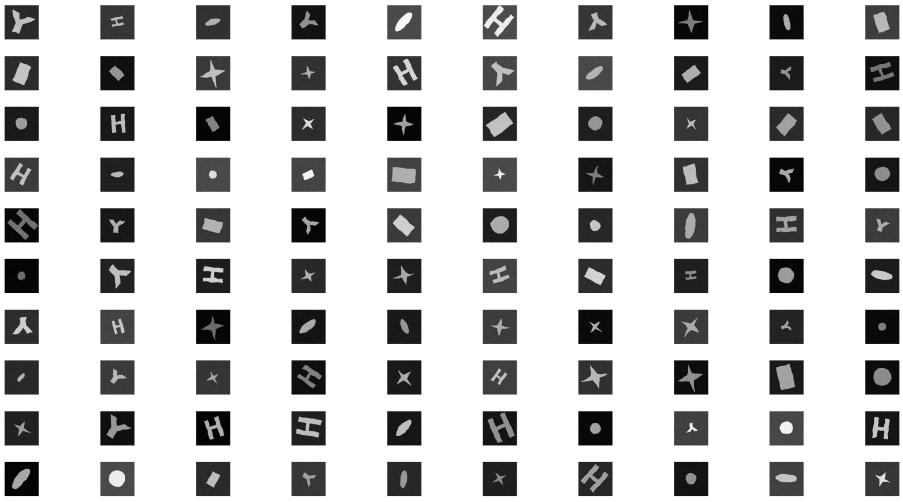


Figure 1

Chaque image généré est accompagnée d'un fichier texte contenant un label qui permet de savoir de quelle forme élémentaire cette image a été créé.

Afin de détecter le contour de la forme présente sur cette image, on va transformer chacune des images en une image binaire ce qui nous permettra de mieux distinguer les contours. Ceci peut être fait de manière simple en utilisant la fonction matlab `imbinarize`.

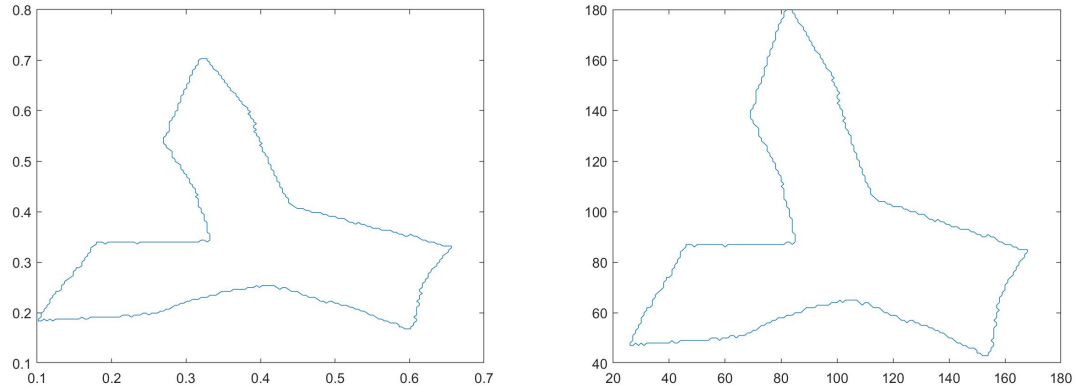
(a) Image originale

(b) Image binarisée

Figure 2

Pour chacune des images obtenus, une détection des contours est faite en utilisant le script `bwboundaries`. Ce script donne une suite de nombres complexes décrivant le contour détectée dans cette image. Ci

dessous un example:



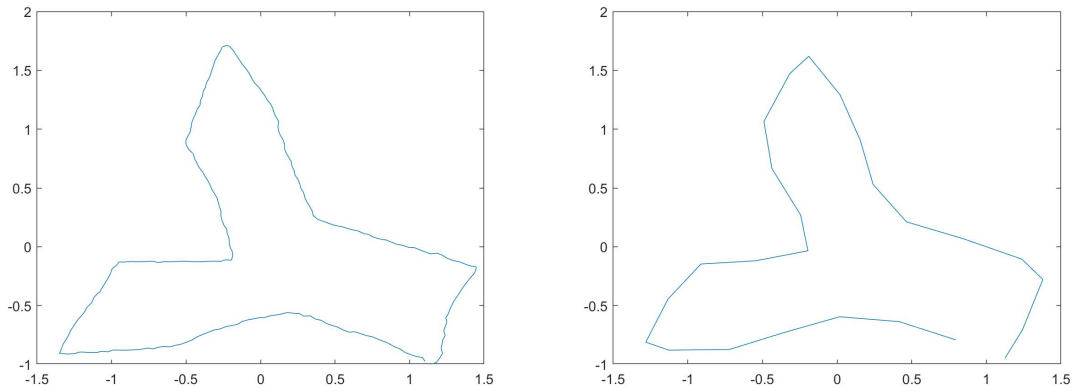
(a) fig 1 (b) fig 2

Figure 3

Transformations élémentaires sur les images

Pour la suite de notre chaine de traitements, on s'intéresse au **descripteurs de Fourier** qui sont obtenus en utilisant la fonction `dfdir`. Cette fonction prend comme arguments un contour et un entier c_{max} qui permet de définir le nombre de coefficients qui seront retournés (qui est égale à $2c_{max} + 1$).

la fonction `dfinv` permet de recréer un contour à partir des coeffecients données, ceci permet de voir à quelle point le contour inital a été déformé et si la forme initale peut encore être distinguée.



(a) $c_{max} = 5$. (b) c_{max} =???.

Figure 4

D'après les figures précédentes, on peut voir que plus c_{max} est élevé plus la reconstruction est fidèle contour originale.

On prend ensuite la valeur absolue des coefficients obtenus pour chaque image. On va par suite créer une matrice C à $2c_{max} + 1$ lignes et 100 colonnes dont chacune des colonnes est égale aux coeffecients obtenus pour l'une des images de notre base.

2 Analyse en composantes principales

Paramètres indépendants

Il a été demandé dans le Tp de trouver des pramaètres qui soient indépondants la translation, l'homothétie et la rotation des images. Les paramètres implémentés sont basés sur une transformation de l'image de sorte à avoir une Bounding Box s'étalant sur $[0,\ 1] \times [0,\ 1]$.

On commence par calculer la taille des segments de la Bounding Box en utilisant la fonction matlab `region_props`, on les note d_x et d_y.

On calcule la surface en cas de Bounding Box unitaire tel que:

$$S_{unitary} = \frac{S_{real}}{d_x d_y} \tag{1}$$

On calcule aussi la position du centroide en cas de Bounding Box unitaire, supposons que la vraie Bounding Box est délimitée par $x_{min}, x_{max}, y_{min}, y_{max}$ et notons le centroide originale c_0 et le centroide unitaire c_1 alors:

$$c_1^x = \frac{c_0^x - x_{min}}{d_x} \tag{2}$$

et

$$c_1^y = \frac{c_0^y - y_{min}}{d_y} \tag{3}$$

Obtention de la base décoréllée

Pour l'obtention de la base décoréllée, on va pour chaque image créer une suite complexe représentant le contour. Cette suite sera donnée comme argument à la fonction `dfdir`, puis la valeur absolue des coefficients obtenus seront pris pour créer la matrice de la base corrélée. Cela étant fait, on utilise la fonction `pca` pour obtenir la base décoréllée.

Tri de la base

La matrice C précédemment obtenue est représentée dont une base dont les vecteurs peuvent être corrélés et redondants. En utilisant une analyse par composantes principales, on obtient une base décorrélée sous forme d'une matrice de taille $(11, 11)$ Pour simplifier et passer à une dimension inférieure, on va par suite trier les colonnes de cette nouvelle base par ordre de variance sachant que les vecteurs ayant la variance la plus élevée permettront de distinguer les formes plus que les autres.

Pour cela on procède de la manière suivante:

- En utilisant la commande `var(C)` on obtient un vecteur ligne dont les éléments correspondent à la variance de chacune des colonnes de C.
- On crée un nouvelle matrice C' ayant 12 lignes et 11 colonnes tel que la première ligne soit égale à `var(C)` et que le deuxième bloque de cette matrice soit égale à C.
- La commande `sortrows` de matlab nous permet de trier les colonnes de C' en prenant comme critère les éléments de la première ligne, puis la seconde etc...
- la matrice C triée est alors égale à $C[2:, :]$.
- En appliquant `var` à la matrice obtenue, on peut voir qu'elle contient les mêmes colonnes que C et que ceux ci sont triés par ordre de variance

Projection sur le s.e.v

Après l'obtention de cette matrice, on va faire en sorte à diminuer la dimension de l'espace de travail en projetant sur un sous espace vectoriel qui est responsable de la majorité de la variance entre les coefficients. Dans la pratique, on va garder les 5 premières colonnes de la matrice triée et les mettre dans une matrice A.

Afin d'obtenir les coefficients dans le sous-espace vectoriel, il faut calculer sa matrice de projection, qu'on peut obtenir en utilisant la formule:

$$P \;=\; A(A^T A)^{-1} A^T \tag{4}$$

A présent, pour chacun des vecteurs x de coefficients correspondant à une image, on obtient son projeté dans le sous-espace vectoriel:

$$x' \;=\; Px \tag{5}$$

Et tout est mis dans une matrice $KM \;=\; [Px_1,\; Px_2,\; ...,\; Px_{100}]$

3 K-moyennes

3.1 Classification

Sachant qu'on a 6 formes élémentaires qui ont servit à la génération des images, on prévoit avoir 6 clusters différents.

La fonction `kmeans` de matlab permet alors d'indexer chacune des colonnes dans un cluster différent ce qui nous donne la classification voulue.

Cependant, comme cet algorithme initialise ses centroides de manière aléatoire, il peut arriver qu'un de ces centroides ne converge pas vers une configuration précise, par example s'il a été initalisé entre deux clusters de même taille. Pour remédier à ce problème, on exécute l'algorithme avec un nombre de centroides supérieur à celui voulu, puis on va regrouper de manière récursive les clusters les plus proches de sorte à n'avoir que 6 à la fin.

Dans la pratique on a choisi d'utiliser `kmeans` avec le double du nombre de clusters réel.

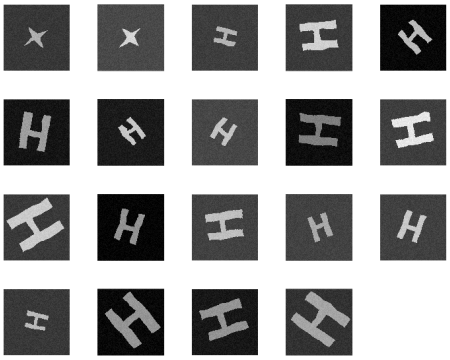
3.2 Prédiction

Pour l'étape de prédiction, après avoir créé un nouvel ensemble d'images d'après notre base, on refait l'analyse en composantes principales sur cette ensemble de sorte à avoir la matrice de coefficients complexes correspondante.

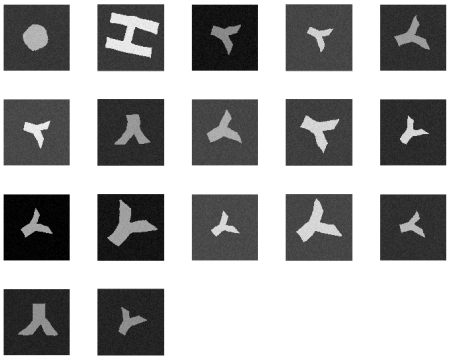
Après, on va calculer pour chaque point la distance quadratique à chacun des centroides et on va lui assigner le centroide le plus proche.

Cela étant fait, un algorithme récursif va calculer la distance entre chacun des centroides et fusionner les deux centroides les plus proches tant que le nombre de centroides présent est différent à 6.

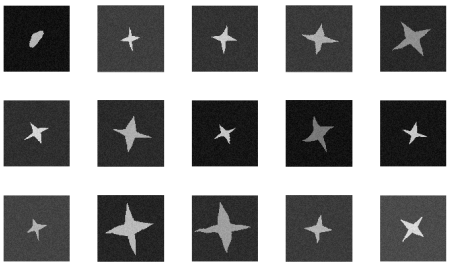
La figure ci-dessous nous permet d'observer le résultat des prédicitons avec 20 coeffecients en guradant les 10 vecteurs ayant la variance la plus élevée de la base fournie par le *pca*.



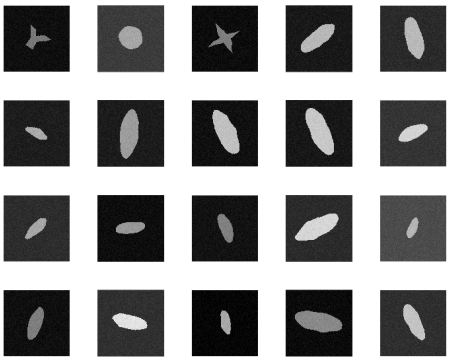
(a) cluster 1



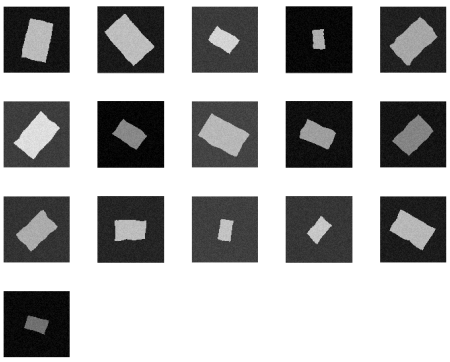
(b) cluster 2



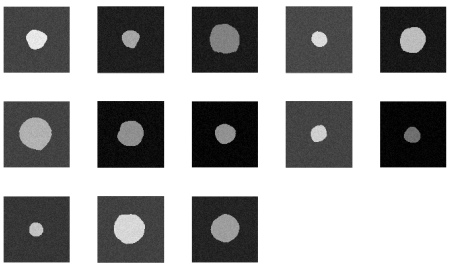
(c) cluster 3



(d) cluster 4



(e) cluster 5



(f) cluster 6

Figure 5: Résultat de la prédiction de `kmeans`.

3.3 Résultats

On obtient un taux d'erreur de 8% pour cette base de données, on remarque que le taux d'erreur ne diminue pas linéairement avec le nombre de clusters utilisé pour la classification mais qu'il existe une zone optimale où les résultats semblent être les meilleurs. Le nombre de colonnes gardés semblent avoir un effet positif mais seulement jusqu'à un certain point. En effet, pour une instance du programme avec 20 coefficients et 10 clusters initiales en obtient 8% dérreur en utilisant tous les vecteurs en sortie du *pca*, ce qui est la même proportion d'erreur qu'on obtient en gardant que les 10 vecteurs ayant la variance la

plus élevée.

Pour comparer le taux dérreur dans des situations différentes, la fonction `calculate_error` a été construite, cette fonction va d'abord identifier les numéros auxquels correspond chacun des clusters en comparant les prédicitons aux fichiers texte de reference. Après le nombre de fausses détections pour chacun des clusters est donné ainsi que le pourcentage dérreur totale de classification.

- Pourcentage d'erreur pour le cluster i:

$$\tau_i \ = \sum \delta_{pred=i,label\neq i} \tag{6}$$

- Pourcentage d'erreur total:

$$\tau = \sum_{clusters} \tau_i \tag{7}$$

3.4 Limitations

1. La détection de contour ne prend en compte que les contours extérieurs, donc la différence entre un disque et un cercle ou la lettre D n'est pas perçu.
2. Sur quelques images, la fonction `dfdir` ne renvoie pas le bon nombre de coefficients, erreur rencontré sur deux images sur $1,67.10^{-3}$ des images générés

4 K plus proches voisins

4.1 Classification

Pour l'entrainement du classificateur, on s'appuie sur le pca présenté dans la section 2. On obtenant notre matrice KM.

Un second paramètre utilisé pour l'entrainement est la liste des classifications, qui, au contraire de la méthode des K-moyennes, doit être fourni. Elle est simplement obtenue en ouvrant chacun des fichier textes correspondant aux images dans le dossier `appr`.

Le classificateur est obtenu en utilisant la fonction `fitcknn` de matlab.

On choisi les paramètres suivants pour notre modèle:

- $c_{max} = 5$ pour la fonction `dfdir`.
- en garde les 7 vecteurs de variance la plus élevée de la base.

4.2 Prédiction

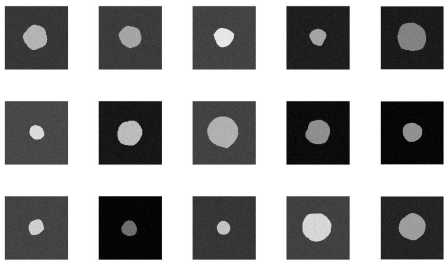
Pour la prédiction, on reconstruit une nouvelle matrice de coefficients de la manière suivante:

On construit la matrice de la projection des coefficients dans le sous espace vectoriel.

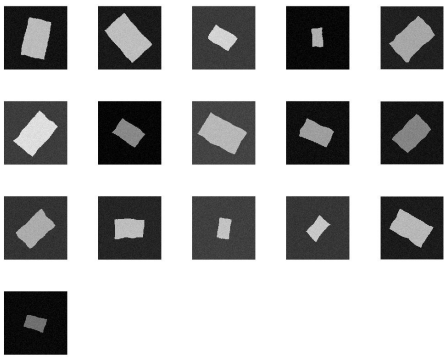
On utilise la fonction `predict` de matlab en lui fournissant le classificateur de 4.1 et la liste des prédictions obtenus.

Le calcul de l'erreur est mené de la même façon que dans le cas des k-moyennes.

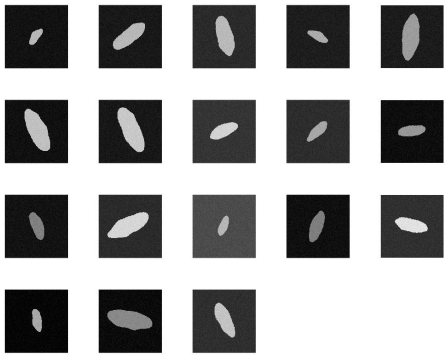
4.3 Résultats



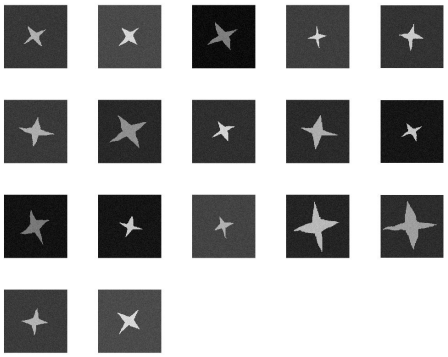
(a) cluster 1



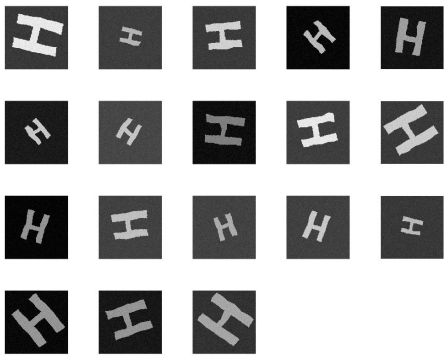
(b) cluster 2



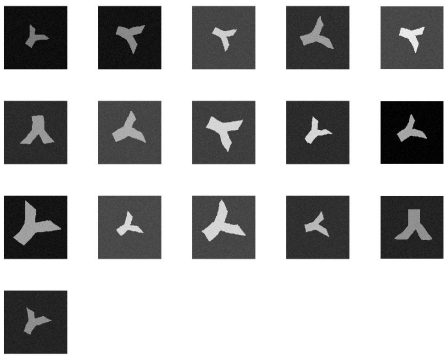
(c) cluster 3



(d) cluster 4



(e) cluster 5



(f) cluster 6

Figure 6: Résultat de la prédiction de `k-nn`.

Comment on peut le remarquer sur la figure précedente, le taux d'erreur obtenu est de 0%.

Dans le cadre de notre étude, la méthode des k voisins les plus proches semble être beaucoup plus précise.

5 Généralisation à d'autres formes

5.1 Modèle

On essaie de tester notre modèle dans le cas ou on travaillent avec d'autres images représentatifs, celles-ci figurent ci-dessous:

(a) cluster 1 (b) cluster 2 (c) cluster 3 (d) cluster 4

(e) cluster 5 (f) cluster 6 (g) cluster 7 (h) cluster 8

(i) cluster 9 (j) cluster 10 (k) cluster 11 (l) cluster 12

(m) cluster 13 (n) cluster 14

Figure 7: Captures de la Transformée de Fourier sur des trames différentes

remarque: Ces lettres ont été choisi en ayant un contour extérieur différent puisque notre algorithme ne différencie pas par exemple entre un 'O' et un disque remplie.

5.2 K-moyennes

Avec le modèle précedemment défini, on crée une base d'entrainemet de 200 images et une base d'images de test de 1000 images. les paramètres utilisés sont les suivants:

20 coeffecients pour la fonction `dfdir`, en garde les 30 vecteurs de variance la plus élevée. Pour `kmeans`, initalement, 28 clusters seront calcués et seront ensuite fusionnéés en 14 pour avoir les prédictions.

Le tableau ci-dessous résume les résultats obtenus.

Centroide	Pourcentage d'erreur
1	0.0135
2	0.0154
3	0
4	0.5035
5	0.0143
6	0.5000
7	0.5000
8	0.000
9	0.4626
10	0.0132
11	0.5000
12	0.000
13	0.5000
14	0.0290
total	3.0515%

5.3 K plus proches voisins

On garde les mêmes paramètres que ceux de 4, les pourcentages d'erreur obtenus sont résumés dans le tableau ci-dessous:

Centroide	Pourcentage d'erreur
1	0.000
2	0.000
3	0.000
4	0.000
5	0.000
6	0.000
7	0.000
8	0.000
9	0.026
10	0.500
11	0.000
12	0.500
13	0.500
14	0.500
total	2.026%

Conclusion

D'après notre étude, on remarque que la méthode des `k-nn` est plus rapide et facile à utiliser en plus d'être beaucoup plus précise (le pourcentage d'erreur est plus faible d'environ 1% alors qu'on a utilisé 4 fois moins de coefficients et de vecteurs dans la base décoréllée.

La méthode de `k-means` ne donne des résultats exploitables qu'en utilisant un nombre assez élevé de coefficients et de vecteurs, en utilisant les mêmes coefficients que pour `k-nn`, en obtient un taux d'erreur d'environ 25%.

Cependant, `kmeans` présente l'avantage de ne pas avoir besoin des vrai descriptions pour l'entrainement de son classificateur ce qui permet son utilisation dans des problèmes d'apprentissage non supervisé.