Travaux pratiques
(6x1h30)

Jean-Luc COLLETTE

(SG6)

Table des matières

1 Caractérisation de formes et classification

1.1 Présentation générale

1.1.1 Introduction

Il est proposé durant ces séances de travaux pratiques, de tester des méthodes de caractérisation des formes, puis d'utiliser ces méthodes dans le but de réaliser une classification.

Préalablement, un traitement sera nécessaire afin d'extraire les formes (images binaires).

Les méthodes de classification qu'on se propose de tester ensuite sont les « k-moyennes » et les « k plus proche voisins » . Elles sont disponibles dans des environnements de calcul comme Python (dans **scikit-learn**) ou Matlab (fonctions **kmeans** et **fitcknn**). Ici, on utilisera Matlab avec les « toolboxes » *Statistics and Machine Learning* et *Image Processing*.

1.1.2 Livrables

1.1.2.1 Compte-rendu

Le compte-rendu fera état des différents essais effectués qui y seront aussi commentés.

1.1.2.2 Programmes

Il est demandé d'écrire des fonctions et des programmes qui soient aussi autonomes que possible, de sorte qu'ils puissent être lancé à nouveau sans problèmes dans une nouvelle session de Matlab et sur un autre poste de travail.

On peut dans cet objectif envisager d'écrire d'une part des fonctions avec les paramètres utiles, et d'autre part des scripts qui lancent ces fonctions avec les paramètres choisis pour la mise en œuvre des différentes méthodes testées.

1.1.2.3 Présentation orale

Une présentation orale de 15 mn conclura les travaux pratiques, durant laquelle une synthèse des méthodes testées sera effectuée.

1.2 Travail à effectuer

1.2.1 Génération d'une base d'images

Après avoir désarchivé le fichier **tp.zip**, ouvrez Matlab puis modifiez éventuellement le répertoire de travail (« folder ») pour lui donner le nom de votre répertoire.

Dans ce répertoire, créez un sous-répertoire **appr** et lancez alors la commande :

```
genere_base
```

Cette commande créera 100 fichiers image mesurexxx.png et 100 fichiers textes mesurexxx.txt dans le répertoire **appr**. Les images crées sont construites à partir de formes initiales stockées dans le répertoire **reference**, et correspondent à des rotations et des homothéties aléatoires appliquées à ces formes, ainsi qu'à de petites déformations de celles-ci.

1.2.2 Visualisation d'une base d'image

Lancez ensuite la commande :

```
visu_base
```

Cette commande permet de visualiser les images préalablement générées, et de voir pour chacune d'elles la classe à laquelle elle appartient (lue dans les fichiers texte).

Le code de cette fonction pourra ensuite être utilisé pour visualiser les différentes étapes des traitements.

1.2.3 Obtention d'une image binaire

Sur l'image en noir et blanc, testez avec la fonction **imbinarize** la génération d'une image binaire. Expliquez et illustrez la méthode utilisée par cette fonction en observant l'histogramme des images avant seuillage.

1.2.4 Extraction du contour de la forme

Avec la fonction **bwboundaries**, générez la suite de nombres complexes qui permettra le calcul des descripteurs de Fourier (voir la fonction **dfdir** en 2.1.1). Reconstruisez un contour à partir de ces descripteurs (voir la function **dfinv** en 2.1.2) pour évaluer qualitativement l'influence du paramètre **cmax**. Pour obtenir des attributs indépendants de la rotation, on prendra ensuite le module des coefficients fournis par la fonction **dfdir**.

1.2.5 Autres paramètres géométriques

Utilisez aussi la fonction **regionprops** sur l'image binaire, afin d'obtenir des paramètres sur les formes sans avoir à extraire un contour. Trouvez des paramètres qui soient indépendants aussi de la translation, la rotation et l'homothétie.

1.2.6 Analyse en composante principale

Les paramètres choisis pour réaliser une classification seront les modules des coefficients obtenus avec la méthode des descripteurs de Fourier. Afin de réduire les éventuelles redondances d'informations, il est possible de réaliser une analyse en composante principale sur ces paramètres. La fonction **pca** de Matlab réalise cette opération. Interprétez le résultat de cette analyse et choisissez le nombre de composantes à conserver pour les étapes suivantes.

1.2.7 Méthode des K-moyennes

Sur la base d'image générée, utilisez la fonction **kmeans** sur les données issues de l'analyse en composantes principales afin de trouver des vecteurs prototypes dans l'espace des paramètres. Représentez ces vecteurs dans l'espace des paramètres et vérifiez leur adéquation avec les données.

1.2.8 Méthode des K plus proches voisins

La fonction **fitcknn** permet de construire facilement une méthode de classification exploitant les données issues de la base d'apprentissage. Les méthodes sont sauvegardées dans des objets de type **ClassificationKNN**. Construisez la méthode exploitant le résultat des k-moyennes (avec K=1, on réalise de la quantification vectorielle) et la méthode exploitant directement les données de la base d'apprentissage.

1.2.9 Evaluation des performances

Créez un répertoire **test** et faites à nouveau appel à la fonction **genere_base** avec les paramètres appropriés. La fonction **predict** permet de mettre en œuvre les méthodes de classification préalablement construites. Lancez les différentes méthodes préalablement construites sur la même base de test et comparez les résultats obtenus.

1.2.10 Généralisation à d'autres formes

Dans le répertoire **reference**, les images initiales ayant servi à générer les différentes bases sont arbitrairement choisies. Proposez d'autres images initiales afin de tester vos programme dans un contexte plus général.

2 Annexes

2.1 Fonctions Matlab fournies

2.1.1 dfdir.m

```
% fonction coeff=dfdir(z,cmax)
% z : suite complexe représentant le contour
% cmax : les coefficients d'indice -cmax à cmax sont conservés
% coeff : tableau des 2*cmax+1 coefficients complexes
function coeff=dfdir(z,cmax)
% on calcule la moyenne
z_moy=mean(z);
N=length(z);
% on calcule les coefficients de Fourier
TC=fft(z-z_moy)/N;
num=(-cmax):cmax;
% on sélectionne les coefficients entre -cmax et cmax
coeff=zeros(2*cmax+1,1);
coeff(end-cmax:end)=TC(1:cmax+1);
coeff(1:cmax)=TC(end-cmax+1:end);
% on retourne la séquence si le parcours est dans le
% sens inverse au sens trigonométrique
if abs(coeff(num==-1))>abs(coeff(num==1))
    coeff=coeff(end:-1:1);
end
% normalisation d'échelle
coeff=coeff/abs(coeff(num==1));
```

2.1.2 dfinv.m

```
% fonction z=dfinv(coeff,N)
% coeff : tableau des 2*cmax+1 oefficients complexes
% N : nombre de points pour le contour reconstruit
% z : suite complexe avec N éléments représentant le contour reconstruit
function z=dfinv(coeff,N)
cmax=(length(coeff)-1)/2;
TC=zeros(N,1);
TC(1:cmax+1)=coeff(end-cmax:end);
TC(end-cmax+1:end)=coeff(1:cmax);
z=ifft(TC)*N;
```

2.1.3 genere_base.m

```
% fonction genere_base(reference,base,N)
% reference : nom du répertoire contenant les formes initiales
% base : nom du répertoire où seront stockées les images
% type_image_ref : type des images de reference
% type_image_base : type des images générées dans la base
% N : nombre d'images générées
function genere_base(reference,base,type_image_ref,type_image_base,N)
if nargin==0
    reference='reference';
    base='appr';
    type_image_ref='bmp';
    type_image_base='png';
    N=100;
end
close all
dos(['del /q .\' base '\*.*']);
sigma=0.05;
DeltaX=22;DeltaY=22;
NX=10;NY=10;
liste=dir(fullfile(reference,['*.' type_image_ref]));
Nfile=length(liste);
for n=1:N
    classe=randi(Nfile,1);
    nom=fullfile(reference,liste(classe).name);
    ima=~imread(nom);
    [L,C]=size(ima);
    rota=pi*rand(1);
    ech=0.4+0.6*rand(1);
    R=[cos(rota) -sin(rota);sin(rota) cos(rota)];
    M=ech*R;
    U0=[C/2;L/2];X0=[C/2;L/2];
    TR=X0-M*U0;
    A=[M TR]';
    T=affine2d(A);
    R=imref2d(size(ima));
    ima=imwarp(ima,T,'cubic','OutputView',R);
    liste_points=cell(NX,NY);
    for nx=1:NX
        for ny=1:NY
            liste_points{nx,ny}=[(nx-1)*DeltaX+(DeltaX/10)*randn(1) ...
                (ny-1)*DeltaY+(DeltaY/10)*randn(1)];
        end
    end
    ima_transf=transf_deform (ima,DeltaX,DeltaY,liste_points);
    ima=zeros(L,C);
    ima(1:size(ima_transf,1),1:size(ima_transf,2))=ima_transf;
    nivmin=0.3*rand(1);
    nivmax=nivmin+0.4+0.3*rand(1);
    mes=(ima<0.5)*nivmin+(ima>=0.5)*nivmax;
    mes=mes+sigma*(1-2*rand(size(mes)));
    imshow(mes),drawnow
    imwrite(mes,fullfile(base,['mesure' sprintf('%03d',n) '.' ...
        type_image_base]));
    fid=fopen(fullfile(base,['mesure' sprintf('%03d',n) '.txt']),'w');
    fprintf(fid,'%d',classe);
    fclose(fid);
end
```

2.1.4 visu_base.m

```
% fonction visu_base(base,arretimage)
% base : nom du répertoire contenant la base d'images
% type_image_base : type des images dans la base
% arretimage : si 0, le défilement est continu, si 1, il faut appuyer sur
% une touche pour obtenir le défilement image par image.
function visu_base(base,type_image_base,arretimage)
if nargin==0
    base='appr';
    type_image_base='png';
    arretimage=0;
end
close all
liste=dir(fullfile(base,['*.' type_image_base]));
set(figure,'Units','normalized','Position',[5 5 90 85]/100)
for n=1:length(liste)
    nom=liste(n).name;
    Y=double(imread(fullfile(base,nom)))/255;
    fid=fopen(fullfile(base,[nom(1:strfind(nom,'.')-1) '.txt']),'r');
    classe=fscanf(fid,'%d');
    fclose (fid);
    subplot(1,2,1)
    imshow(Y),title(['fichier ' nom ', classe ' int2str(classe)]),drawnow
    subplot(1,2,2)
    % on peut ici rajouter le code associé à différents traitements
    imshow(Y),title('image noir et blanc'),drawnow
    if arretimage
        pause()
    end
end
```

2.2 Descripteurs de Fourier normalisés

2.2.1 Représentation d'un contour

Avec une liste ordonnée de N points (x_m, y_m) représentant un contour fermé, on peut construire une suite de nombres complexes :

$$z_m = x_m + jy_m$$

En considérant que cette suite est la représentation sur une période d'un signal périodique de période N, on peut lui associer une décomposition en série de Fourier :

$$a_k = \frac{1}{N}\sum_{m=0}^{N-1}(z_m - \bar{z})\exp\left(-\frac{2\pi jkm}{N}\right)$$

La valeur $\bar{z}$ est la moyenne du signal complexe.

Pour approximer le contour, on peut ne conserver dans cette décomposition que les coefficients les plus significatifs, pour $k_{min} \leq k \leq k_{max}$. L'approximation du contour est alors donnée par le signal complexe :

$$\hat{z}_m = \bar{z} + \sum_{k=k_{min}}^{k_{max}} a_k \exp\left(\frac{2\pi jkm}{N}\right)$$

Les attributs caractérisant la forme, invariants par translation et par rotation peuvent être alors les $d_k = |a_k|$ avec $d_0 = 0$ (si le contour fermé est parcouru dans le sens trigonométrique).

2.2.2 Normalisations possibles

2.2.2.1 Invariance vis-à-vis du sens de parcours du contour

En fonction du sens de parcours du contour, on obtient des coefficients a_k pour le signal z_m ou les coefficients a_{-k} pour le signal $z_{(-m)[N]}$. Pour assurer l'unicité de la représentation, il convient de réaliser la permutation ci-dessous dans le cas où $d_1 < d_{-1}$.

$$a_{temp} = a_k$$

$$a_k = a_{-k}$$

$$a_{-k} = a_{temp}$$

Il faut dans ces conditions que $k_{min} = -k_{max}$ afin d'obtenir des vecteurs qui puissent être comparés entre eux, avec ou sans permutation.

2.2.2.2 Invariance vis-à-vis de l'échelle

Avec la permutation précédente, d_1 a la valeur la plus grande. En divisant tous les coefficients a_k par d_1, on obtient des coefficients indépendants du facteur d'échelle.

2.2.2.3 Coefficients complexes normalisés

Cette dernière normalisation n'est pas utilisée dans le cadre de ce TP. Elle est juste décrite à titre d'information.

L'inconvénient de l'utilisation des paramètres d_k est qu'une infinité de formes différentes peuvent être associées au même vecteur de paramètres d_k (en ne modifiant que la phase des coefficients a_k).

Pour caractériser la forme de façon plus pertinente, il convient donc de revenir aux coefficients complexes a_k initialement obtenus par décomposition en série de Fourier. Pour qu'ils soient indépendants aussi de la rotation, comme les d_k, il convient de procéder à des modifications sur les a_k en appliquant les quatre équations ci-dessous.

$$\phi = \frac{\mathrm{Arg}\{a_{-1}a_1\}}{2}$$

$$a_k = a_k \exp(-j\phi)$$

$$\theta = \mathrm{Arg}\{a_1\}$$

$$a_k = a_k \exp(-j\theta k)$$

En effet, si on obtient pour une forme donnée des coefficients a_k, la même forme orientée autrement avec une origine différente pour le signal complexe représentant le contour, sera associée à des coefficients de la forme $a_k' = a_k \exp(j\alpha + j\beta k)$. L'angle α varie selon l'orientation de la forme et l'angle β varie selon l'origine du signal complexe (le décalage d'origine correspond à un retard pur, induisant une variation linéaire de la phase).

Ainsi, $\mathrm{Arg}\{a_1'a_{-1}'\} = \mathrm{Arg}\{a_1a_{-1}\} + 2\alpha$ pour toute valeurs de β. Avec les deux premières équations faisant intervenir le paramètre ϕ, on obtient donc des coefficients qui ne dépendent plus de l'angle d'orientation de la forme (à une valeur π près).

Cette première correction étant faite, il reste à obtenir des coefficients associés à la même origine du signal complexe. On fait appel alors aux deux dernières équations faisant intervenir le paramètre θ qui rajoutent alors à l'argument des coefficients un terme linéaire qui fait que le coefficient complexe a_1 devienne réel. En conséquence, le coefficient a_{-1} devient lui aussi réel.