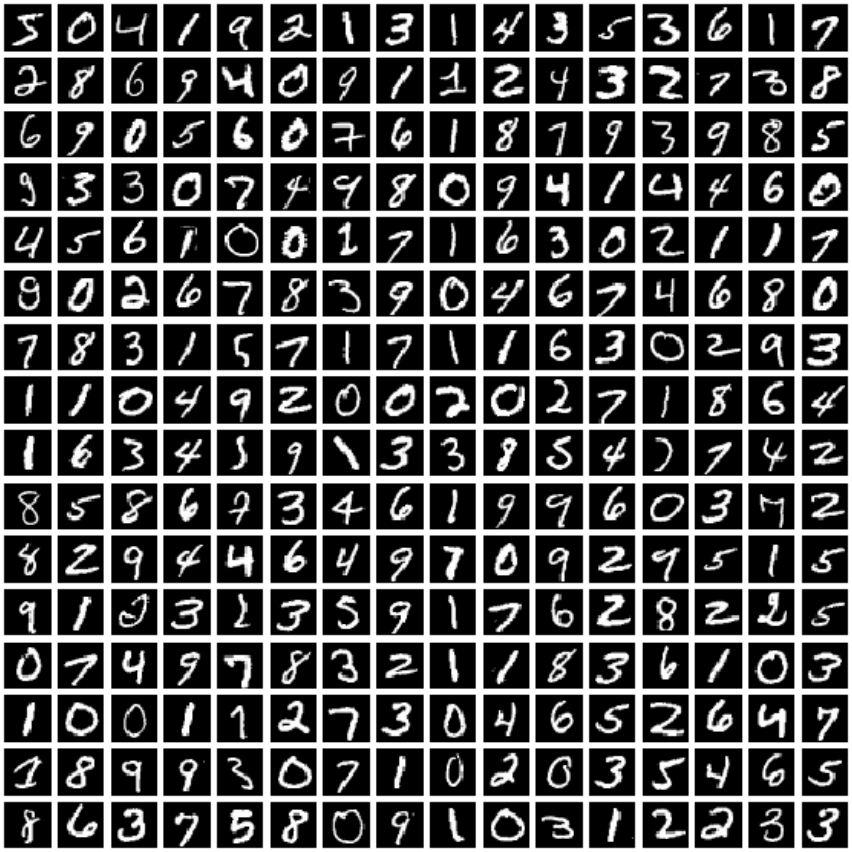


Rapport TP IN54

Analyse et reconnaissance de chiffres manuscrits

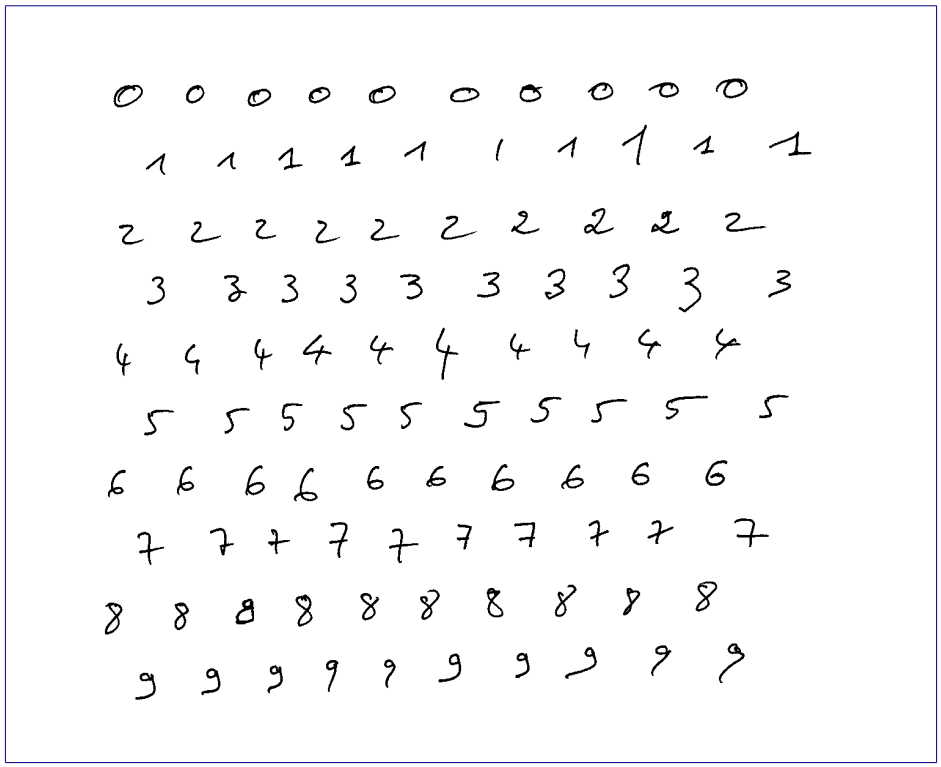
Ackermann Audric | IN54 - UTBM | 24/11/13

Introduction

Objectif du TP

L'objectif de ce TP est la reconnaissance de chiffres manuscrits contenus dans une image, grâce à l'utilisation de Matlab.

Nous utiliserons pour cela différents classifieurs, et les combineront afin d'améliorer les résultats de chacun indépendamment. L'image que nous avons est une image contenant tout les chiffres en lignes, comme celle-ci :



La première étape est découper l'image afin d'isoler chaque chiffres.

Découpe de l'image

Via regionProps

La première approche est l'utilisation de la méthode regionProps de Matlab. Elle permet d'analyser une image et, suivant les paramètres qu'on lui donne, d'extraire les BoundingBox (boîte englobante) des chiffres. Pour cela, l'appel nécessaire est simplement :

```
bb = regionprops(BW, {'BoundingBox'});
```

On obtient ainsi une liste dont on peut extraire les bounding box seules via :

```
boundingboxes = cat(1, bb.BoundingBox);
```

Cette méthode marche, mais elle ne respectait pas l'énoncé du sujet. Nous ne la détaillons donc pas plus ici.

Via la projection

Cette méthode effectue la séparation des chiffres en recherchant premièrement les indices de début et de fin des lignes de chiffre dans l'image. Pour ce faire, la méthode proposée est la suivante :

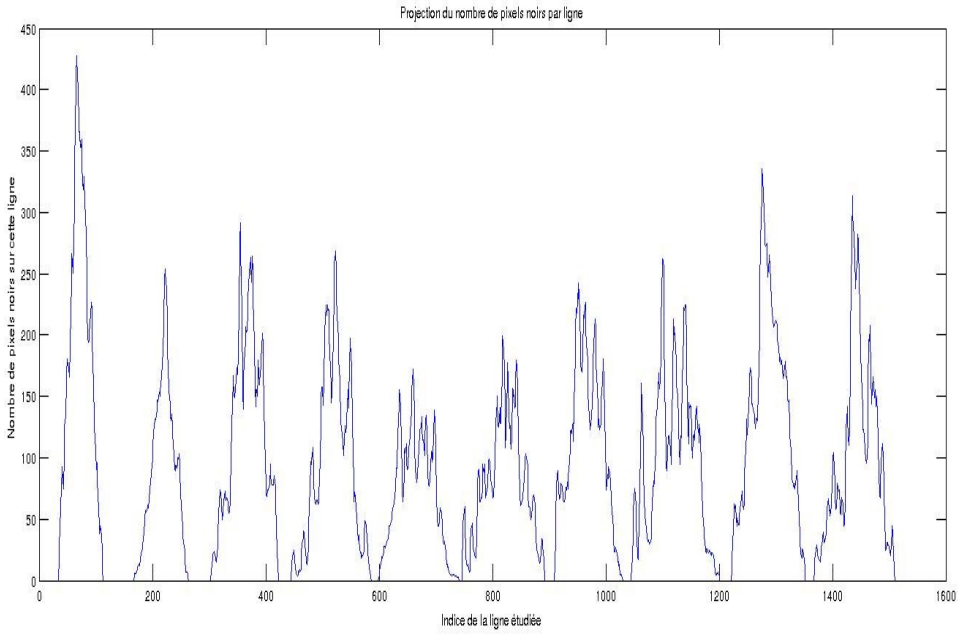
- On somme ligne à ligne (de pixels) le nombre de pixels noirs. On obtient ainsi une liste contenant le nombre de pixels noirs de chaque ligne. Une ligne contenant 0 pixels noirs est une ligne sans chiffres donc.
- On cherche alors les lignes où l'on passe d' un nombre de pixels noirs égal à 0 à un > 0. Cela nous donne les indices de début d'une ligne.
- On fait de même pour trouver les positions des fin de lignes de chiffres manuscrits.

Nous obtenons alors deux indices par ligne qui nous permettent d'extraire une sous-image de l'image de départ comme celle-ci :

Afin de réutiliser les composants existants de Matlab, nous utilisons la fonction :

```
imageData=sum(BW>0,2);
```

Cette fonction effectue la somme des pixels noirs de l'image, en la traitant comme une matrice à 2 dimensions, donc lignes par lignes. On obtient ainsi dans imageData une liste des nombres de pixels noirs lignes par lignes. En voici un exemple :



Nous pouvons y voir les 10 lignes de chiffres de l'image.

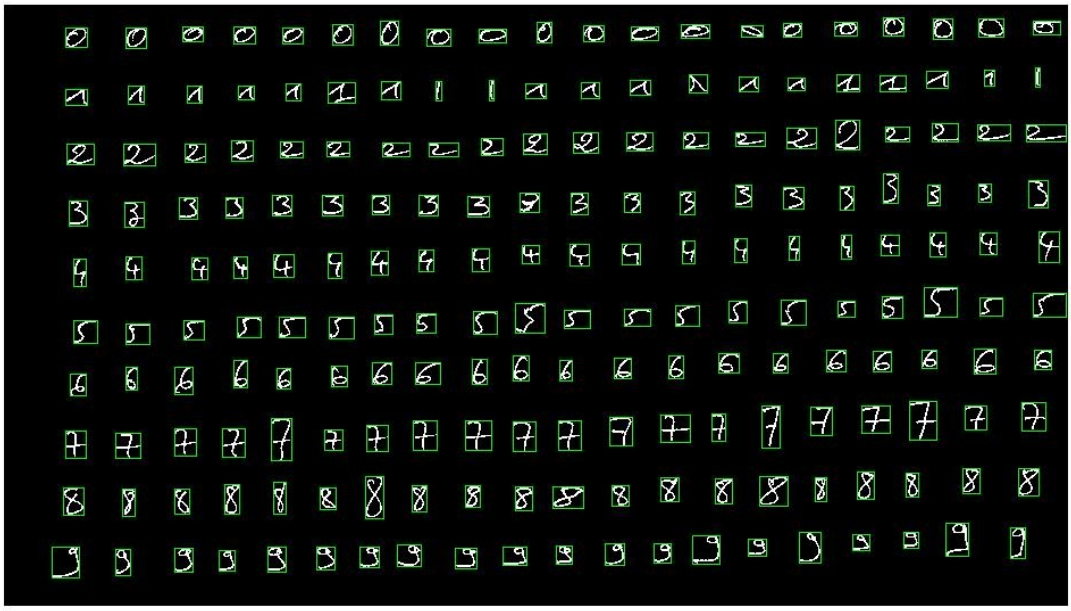
Il nous suffit alors d'extraire les passages ou la valeur change de 0 vers autre chose. Pour cela, on cherche dans l'image le premier front montant et à partir de celui-ci le premier retour à 0. On a alors les deux indices de la première ligne de chiffre.

Pour les suivantes, il suffit de recommencer la recherche sur la partie restante des données, en y ajoutant l'indice de départ comme suit :

```
1. edge=find(imageData(lastStop:end)==0, 1, 'first') + lastStop;
2. fallingEdge=find(imageData(edge:end), 1, 'first') + edge;
```

En ajoutant cela à une liste, on obtient la liste des indices de début et de fin des lignes de chiffres.

Afin de vérifier le bon fonctionnement avant de continuer, nous avons développé une fonction permettant d'afficher les bounding boxes trouvées en overlay sur l'image d'origine. En voici le résultat :



Nous avons également vérifié plus précisément, en extrayant l'image associée à chaque bounding boxes dans l'image et en vérifiant que le chiffre entier y est bien, avec un pixel de chaque coté.

Structures de données utilisées

La structure de données permettant de stocker les bounding boxes des chiffres est simple : c'est une liste de 4 éléments toute simple.

- le premier correspond au x du coin supérieur gauche
- le second au y de ce même coin
- le troisième correspond au x du coin inférieur droit
- et le dernier au y de ce même coin

Afin de stocker l'ensemble des rectangles englobant, il faut donc avoir une liste de toutes les listes ci dessus par chiffre. On obtient donc une matrice de :

'nombre de chiffre dans l'image' x 4 (car 4 coordonnées par chiffre)

Limites

Cette méthode à l'avantage d'être facile à mettre en place, mais elle a ses limites.

Premièrement si les lignes de chiffres ne sont pas toutes séparées par au moins un pixels noir, on arrivera pas à détecter le front montant (en tout cas pas avec l'implémentation actuelle). Celui-ci est basé sur le passage d'un nombre de pixel blanc par ligne de 0 à quelques chose >0. Si on ne repasse pas par 0 entre deux ligne, l'extracteur de ligne n'en comptera qu'une.

Deuxièmement, si les chiffres ne sont pas correctement orientés, l'extraction risque d'extraire de nombreux pixels en trop. Imaginons le chiffre suivant :

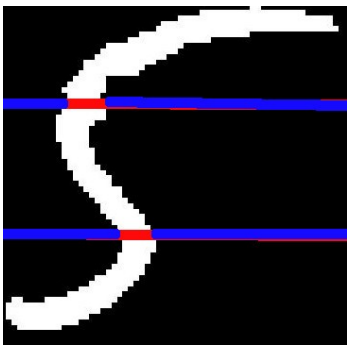
L'analyse des lignes et colonnes de chiffres nous sortira le rectangle englobant de ce chiffre, mais il serait bien plus utile de remettre ce chiffre dans son axe horizontal. Ses données nous seraient effectivement bien plus utiles si ce chiffre était droit.

Une fois l'extraction des chiffres validée, on passe à la classification de ceux-ci.

Classifieur par profils gauche/droits

Principe

Cette méthode se base sur l'étude des « profils » des chiffres. Pour faire simple, nous découpons les chiffres par un certains nombre de lignes horizontales, et nous mesurons alors à partir de la droite et de la gauche la distance du bord au début du chiffre. Ainsi, pour le chiffre 5 et un nombre de lignes de 2 on pourrait avoir :



Les lignes bleues étant les profils gauche et droit du chiffre.

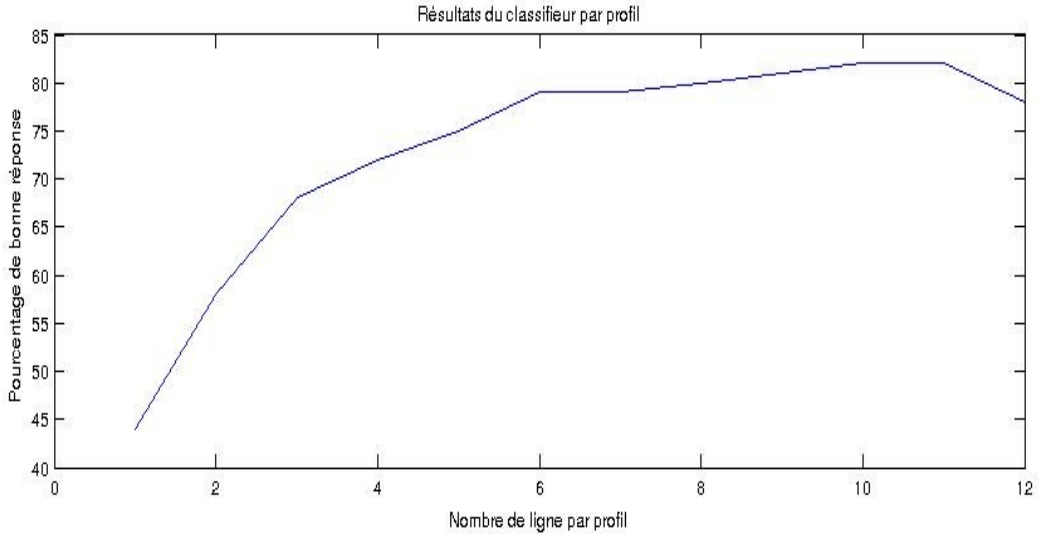
Cette analyse du chiffre nous permet d'obtenir un vecteur moyen pour l'ensemble des chiffres 5 de la base d'apprentissage. Et de même pour les autres chiffres. A partir de là nous avons donc un vecteur moyen de profils pour chaque classe de chiffre. Par classe nous entendons 1,2,...9.

A partir de là il nous suffit, pour chaque chiffre de l'image de test, de sélectionner la classe la plus proche de notre chiffre parmi tout les chiffres. La distance utilisée est la distance euclidienne.

Tests et évaluation

Afin de déterminer le nombre de ligne par profils idéal il a fallut expérimenter. Pour cela rien de plus simple. On met le bloc d'exécution dans une boucle faisant varier le nombre de ligne par profil. Les résultats sont ci-dessous.

Résultats et analyse



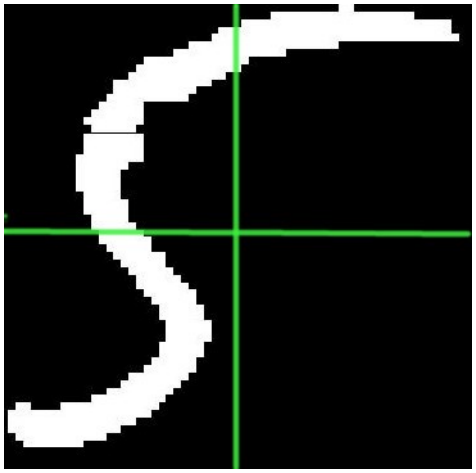
C'est apparemment meilleur que ce que l'on ai sensé trouvé avec ce classifieur. J'ai pourtant fait comme demandé. La seule chose que j'ai changé par rapport à l'énoncé c'est que j'ai rajouté un deuxième pixel blanc autour du chiffre lors de l'extraction, car je n'en voyais lorsqu'il n'y en avait qu'un. On voit sur ce graphique que les valeurs tournent autour de 80 %, avec un pic en 10-11 et une valeur de 82. Après 12, les valeurs continuent de descendre. On peut expliquer comme ceci :

- Au départ, il n'y a pas assez de profils pour isoler clairement un chiffre d'une classe ou de l'autre, les classes sont trop proches les unes des autre, par exemple 3 et 2 avec une seule ligne.
- Après, on augmente car on arrive de mieux en mieux à les séparer.
- Puis on rechute car on a une trop grande spécialisation : les lignes seront trop nombreuses, et on correspondra plus ou moins à toutes les classes.

Classifieur par densité et KPPV

Principe

Ce classifieur se base sur la méthode KPPV (K plus proches voisins). On découpe de la même façon les chiffres dans l'image, mais cette fois on calcul les densité de pixels blancs du chiffre dans un certain nombre de zones. En voici un exemple pour le chiffre 5 et un nombres de zones horizontales et verticales de 2.



On compte donc le nombre de pixels blancs dans chaque zones, que l'on divise par le nombre de pixels total dans cette zones afin d'avoir une densité de pixels blancs, par zones. Ici, nous ne calculons pas de vecteur moyen par classe, nous gardons la totalité des analyses des chiffres de l'image d'apprentissage. Nous recommençons alors la même opération en analysant les chiffres de l'image de test.

Ensuite, on applique la méthode des k plus proches voisins (KPPV). Elle consiste à déterminer les k voisins les plus proches du chiffres que l'on cherche à classifier. Ainsi, on commence par créer une liste des voisins les plus proches du chiffre en question, puis on compte le détermine le pourcentage de fois que c'est le 1, puis le 2, et ainsi de suite. On obtient alors une liste de probabilités que le chiffre en question appartienne à chaque classe.

Structures de données utilisées

La structure de donnée utilisée pour stocker les densités des chiffres de l'image est une matrice 3 dimensions. La première est de taille le nombre de chiffres dans l'image. La seconde et la troisième servent à stocker les densités par chiffre proprement dites. Ainsi, les deux dernières forment une matrice de 'nombre de zones verticales' x 'nombre de zones horizontales'. On y stocke donc les valeurs des densités pour un chiffre en particulier, pour une zone horizontale précise et une zone verticale précise.

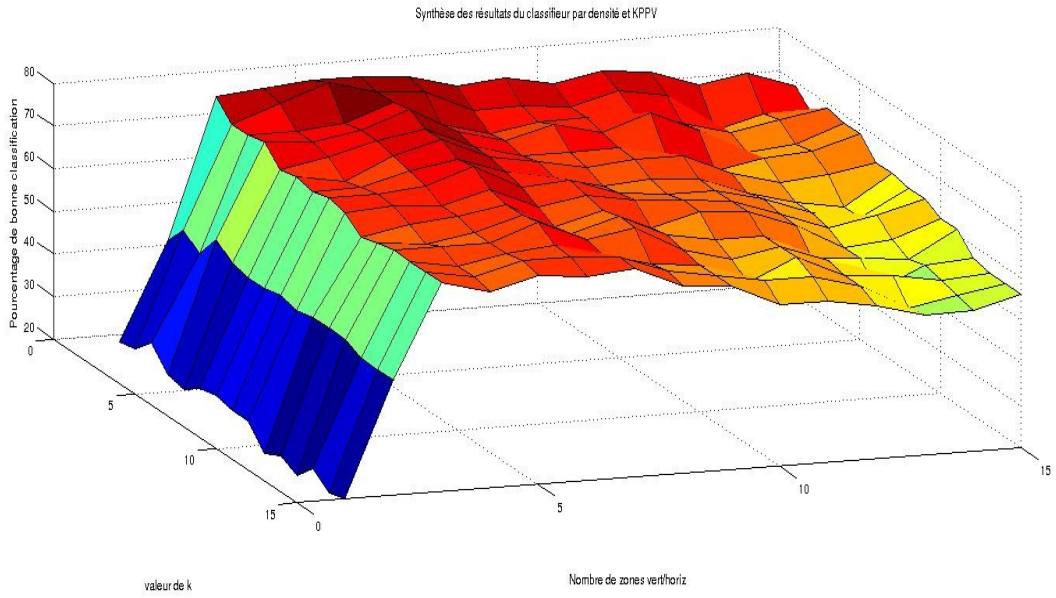
Tests et évaluation

Les tests que l'on peut réaliser sur ce classifieur sont de faire varier k, ou encore le nombre de zones verticales ou horizontales.

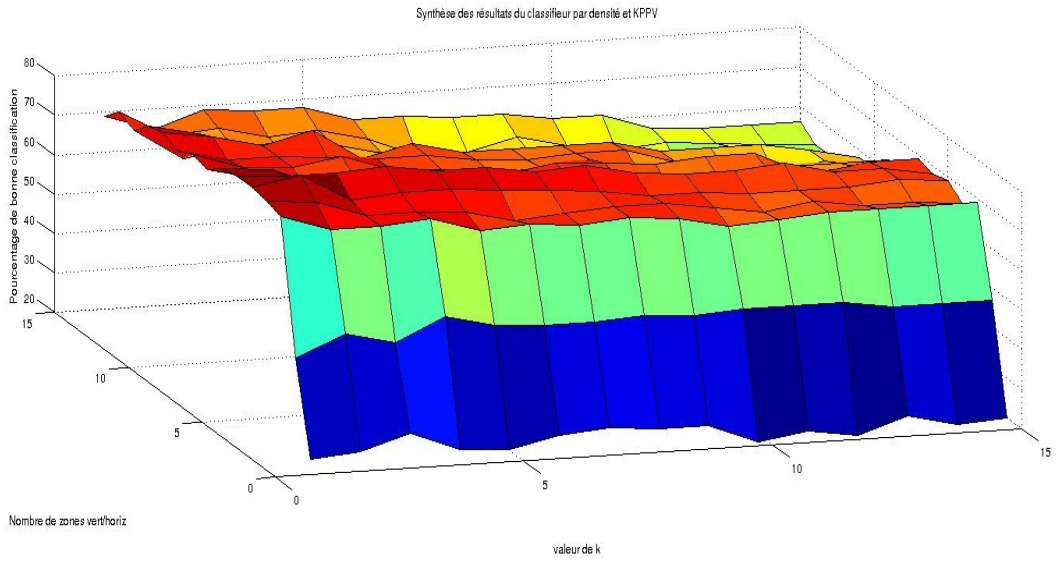
Résultats et analyse

Voici les différentes courbes lorsque l'on fait varier les différents paramètres.

Variation du nombre de zones horizontales/verticales :



Variation de la valeur de k :



Les résultats ci-dessus sont les tests pour un nombre de zones de 1 à 15 et un k variant de 1 à 15. Plus la valeur est rouge foncée, meilleur est la reconnaissance des chiffres. La **valeur maximale** est atteinte pour k=2 et un nombre de zones de 5, soit **81 %**. Sur le deuxième, on voit plus clairement l'influence de k :

- Si k=1, on ne retient que le voisin le plus proche. L'intérêt de la méthode KPPV est donc limitée dans ce cas.
- Si k>2, la classification donne des résultats contenant plus de classes, les résultats sont donc plus éparpillés dans l'espace de solutions possibles (les classes de chiffres). Il devient alors difficile de distinguer la classe réelle de ce chiffre dans cet ensemble.

Concernant l'influence du nombre de zones (le graphique 1 est plus lisible pour ce cas):

- Si le nombre de zones est trop faible (<3), les densités de chacune représentent une trop grande surface du chiffre, et des incertitudes apparaissent d'une classe à l'autre. En effet, le chiffre 1 et 7, risque d'avoir des densités de zones assez proches, et donc le classifieur se trompera.
- Si le nombre de zones est trop important (>5), la reconnaissance devient un peu moins bonne. C'est essentiellement du au fait que le classifieur va séparer en zones très petites, et que leurs densité si précises ne voudront finalement plus rien dire. En effet, en faisant des zones trop petites, on risque de se rapprocher plus du 3 ou du 8 pour un 9 dans une bonne partie des cas. Bien sûr cela provient également de nos données d'apprentissage.

Combinaisons de classifieurs

Principe

La combinaison des classifieurs est l'étape où l'on réutilise les résultats des deux précédents classifieurs afin d'obtenir une meilleure classification. Pour ce faire, on lit les données sauvées dans les fichiers lors des l'exécution des deux classifieurs (avec leurs meilleures paramètres) grâce à la fonction :

```
importdata('filename')
```

On combine ces deux classifieurs avec ou leur addition, ou leur produit. Ceci afin de déterminer laquelle des deux méthodes est la meilleure.

Résultats

En guise de rappel, voici les résultats des deux classifieurs indépendamment l'un de l'autre (avec les meilleurs paramètres de chacun) :

- Classifieur par profil + distance minimale : 82 %
- Classifieur par densité + KPPV : 81 %

Et voici les résultats des différentes combinaisons :

- Par addition : 88 %
- Par produit : 87%

Bien que cela soit tout de même 5 points au dessus du meilleur des classifieurs, on aurait pu s'attendre à mieux. On peut quand même noter que c'est une amélioration, donc la combinaison des deux classifieurs tend à compenser les erreurs de l'un par les réussites de l'autre. Par contre, nous il n'y a pas de différences notable entre la méthode par addition ou produit. Cela vient certainement de la proximité isolément déjà des deux classifieurs. La faiblesse de l'amélioration vient peut être de la base de test qui n'est pas très complète : seulement 20 chiffres de chaque classe et tous écris par la même personne.

Conclusion

Développements réalisés

Afin de réaliser ce TP, une certaine quantité de développement à dû être fournis. On peut notamment citer les fonctions demandées afin de produire l'analyse par profils ou densité, et toutes celles qui vont avec. Il a également fallut développer certaines fonctions spécifiquement pour ce rapport, notamment pour le graphique des résultats du KPPV qui est un graphique 3D et dont la création demande certaines adaptations vis à vis des structures existantes dans le programme. La seule chose qui n'a pas été développé a été l'utilisation d'autres distances dans le classifieur 1, ainsi que de programmer d'autres opérateurs de combinaisons, mais ils faisaient tout les deux parties de la partie facultative du TP.

Résultats obtenus

Ce TP nous permet donc de reconnaître un chiffre manuscrit dans une image avec une précision de 87 % au mieux. C'est plutôt bon pour un TP d'une vingtaine d'heures.

Bien sûr, ces classifieurs ne sont pas près à affronter un cas « réel » où l'on aurait par exemple, des chiffres écris par plusieurs personnes, dans des orientations pas idéales ou dans des situations variées (écritures penchées, donc lignes de chiffre non horizontale). Il faudrait alors nécessairement leur fournir une base d'apprentissage plus grande.

Améliorations possibles

Afin d'améliorer les résultats des différents classifieurs, voici une liste d'améliorations possibles :

- Ajouter des lignes verticales pour l'analyse de profils. Cela permettrait de gagner peut être des points précieux sur le classifieur par profil
- Analyser les résultats de chacun des classifieurs par classe de chiffre plutôt que sur le résultat global. Cela permettrait de définir sur quels chiffres tel classifieur est bon ainsi que ceux où il est moins bon.

Perspectives envisageables

Nombre de perspectives sont envisageables pour ce TP. On pourrait y ajouter la reconnaissance des chiffres romains, des écritures non attachées, …. Il pourrait également servir à d'autres TP ou projet de squelette, puisque l'ajout de nouveaux classifieurs peut se faire assez facilement, à partir du moment où l'on respecte le format des fichiers de sauvegarde des probabilités.

Outre le fait que ce TP m'a énormément appris sur l'utilisation de Matlab (que j'ai découvert pour l'occasion), il m'a également appris une méthodologie de classification de formes, ce qui est tout de même l'intitulé de l'uv. L'utilisation de différents classifieurs et le

besoin d'expérimenter les valeurs des différents paramètres permettant d'obtenir un résultat maximal me permet aujourd'hui d'y voir plus clair.