RAPPORT TP IN54 A2020

Reconnaissance de chiffres manuscrits

Daniel DI DIO BALSAMO - Elom KOSSI

Table des matières

[1 Introduction 2](#_Toc58676304)

[1.1 Objectifs 2](#_Toc58676305)

[1.2 Remarque sur les scripts 2](#_Toc58676306)

[2 Localisation et extraction des chiffres 2](#_Toc58676307)

[3 Classification par distance euclidienne minimum 2](#_Toc58676308)

[3.1 Le vecteur de caractéristiques 2](#_Toc58676309)

[3.2 Apprentissage du classifieur 2](#_Toc58676310)

[3.3 Décision 2](#_Toc58676311)

[3.4 Analyse des résultats 2](#_Toc58676312)

[4 Classification par densités et KPPV 2](#_Toc58676313)

[4.1 Principe et Calcul des densités 2](#_Toc58676314)

[4.2 Apprentissage du classifieur 2](#_Toc58676315)

[4.3 Décision 2](#_Toc58676316)

[4.4 Analyse des résultats 2](#_Toc58676317)

[5 Combinaison de classifieurs 2](#_Toc58676318)

[6 Conclusion 2](#_Toc58676319)

# **Introduction**

## Objectifs

Le but de ce TP est de mettre en place un système complet de reconnaissance de chiffres manuscrits avec Octave.

La reconnaissance se fera via deux méthodes : classification par distance euclidienne minimale et classification par densités et KPPV.

Nous traiterons tout d’abord le sujet de l’extraction des chiffres des images, puis pour chacune des méthodes nous verrons dans un premier temps son fonctionnement, puis les résultats obtenus seront analysés afin de déterminer son paramétrage idéal et de juger de son efficacité.

Nous verrons ensuite comment améliorer la reconnaissance en combinant les résultats de deux classifieurs.

Enfin, nous terminerons par une discussion autour de la pertinence de ce système de reconnaissance de chiffres, et de l’influence de la taille des bases de d’apprentissage et de test.

## Remarque sur les scripts

Même si tout sera détaillé au fur et à mesure, précisons quelques éléments quant à l’utilisation des scripts.

Le package « *image* » doit être chargé avant de lancer les scripts avec la commande: « *pkg load image* ».

Pour tester une configuration particulière lancer les scripts suivants:

* *main* : tester un paramétrage particulier pour les 2 classifieurs et la combinaison.
* *launchEuclidian* : tester un paramétrage particulier pour le classifieur par distance euclidienne minimale seulement.
* *launchKNN* : tester un paramétrage particulier pour le classifieur par densités et KPPV seulement.

Pour analyser les performances d’un classifieur et afficher les courbes associées, lancer les scripts suivants :

* *benchmarkEuclidian* : réaliser un benchmark du classifieur par distance euclidienne minimale et afficher les courbes des taux de reconnaissance associées.
* *benchmarkKNN* : réaliser un benchmark du classifieur par densités et KPPV et afficher les courbes des taux de reconnaissance associées.

# **Localisation et extraction des chiffres**

La première étape de ce système est de localiser et extraire les chiffres des images.

Pour cela, on génère les histogrammes de projections horizontales/verticales, en comptant le nombre de pixels noirs par lignes/colonnes.

Une image contenant cintre

Description générée automatiquement

Les deux histogrammes ainsi obtenus consistent en une série de pics.

On détermine alors le début et la fin de chaque plage sur les deux histogrammes, ce qui donne une première délimitation des chiffres.

Cependant, on cherche à obtenir des rectangles englobants tels qu’il n’existe aucune ligne/colonne ne contenant aucun pixel noir, ce qui serait le cas avec cette première délimitation.

Il faut donc répéter le procédé précédant à chaque rectangle obtenu pour obtenir des rectangles englobants ajustés aux dimensions des chiffres.

La figure suivante affiche les rectangles ajustés sur les images « *app.tif* »

et « *test.tif* » respectivement, avec le script « *showRectangle.m* » :

Une image contenant texte

Description générée automatiquement

Figure 1: Rectangles ajustés de « app.tif »

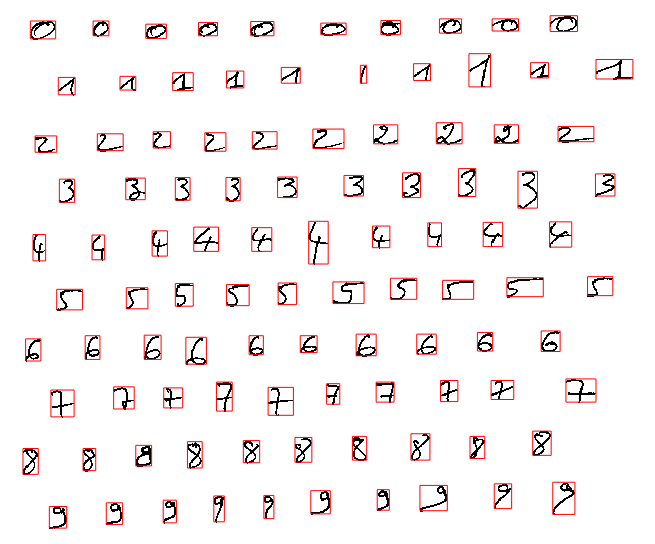


Figure 2: Rectangles ajustées de « test.tif »

Les scripts *getColumns*, *getLines* et *buildRectangle* permettent de réaliser ces opérations.

Il reste finalement à stocker les caractéristiques de chaque rectangle afin de pouvoir traiter les chiffres par la suite.

Le vecteur contenant les rectangles a pour composantes :

* Numéro 1 : ligne du coin en haut à gauche du rectangle.
* Numéro 2 : colonne du coin en haut gauche du rectangle.
* Numéro 3 : largeur du rectangle.
* Numéro 4 : longueur du rectangle.

Cette méthode d’extraction des chiffres est très intéressante car elle permet d’extraire les chiffres en un temps raisonnable en utilisant seulement des histogrammes.

Cependant, les chiffres doivent malgré tout être ordonnés selon des lignes/colonnes même si elles ne sont pas parfaitement alignés.

En d’autres termes, cette méthode ne fonctionnerait pas pour une image où les chiffres auraient été placés aléatoirement (et sans collisions) sur l’image.

# **Classification par distance euclidienne minimum**

## Le vecteur de caractéristiques

Ce premier classifieur se base sur la répartition des pixels selon les lignes, en déterminant le profil droit et le profil gauche de chaque chiffre.

Tout d'abord, on choisit uniformément d lignes dans le rectangle englobant du chiffre étudié.

Pour chacune de ces lignes, on calcule la distance entre le bord gauche et le premier pixel noir de cette ligne, et on obtient ainsi le profil gauche du chiffre.

On détermine également le profil droit du chiffre en calculant la distance entre le bord droit et le premier pixel noir pour chaque ligne.



On obtient ainsi un vecteur de caractéristique de d composantes, qui est la concaténation du profil gauche et droit du chiffre.

On normalise finalement ce vecteur en le divisant par la largeur du rectangle englobant, afin de tenir compte du fait que tous les chiffres ne font pas la même largeur.

Le script « *extractProfile.m* » réalise cette opération.

## Apprentissage du classifieur

On souhaite à présent calculer le centre de chaque classe.

Pour cela, on détermine le vecteur de caractéristiques de chaque chiffre de l'image « *app.tif* » (image d'entraînement), c’est-à-dire que l’on détermine le profil de chaque chiffre.

Ensuite, on calcule le vecteur moyen des vecteurs de caractéristiques correspondant au chiffre de la classe (20 dans ce cas).

Le vecteur contenant les centres est de dimension 10 x d, puisque l'on considère 10 chiffres pour lesquels on a fait la moyenne des vecteurs de caractéristiques de d composantes.

Ces centres permettent de caractériser chaque classe, et servira de référence pour la classification.

Le script « computeCenters.m » calcule les centres qui sont sauvegardés dans « *euclidianCenters.mat* »

## Décision

Maintenant que les centres de classes sont disponibles, on peut déterminer le vecteur de probabilités caractérisant la probabilité que les chiffres de l'image d'entraînement « test.tif » appartienne à chacune des classes.

Pour cela, le vecteur de caractéristiques de chaque chiffre est calculé puis comparé avec le centre de chaque classe de la manière suivante :

* : Vecteur de caractéristiques du chiffre à classer (profil)
* : Centre de la classe i

Cette fonction, *softmax*, permet d'accentuer les différences et assure d'avoir des résultats dont la décision est « franche ».

Le chiffre reconnu par le classifieur est celui correspondant à la classe ayant la probabilité la plus élevée.

Pour calculer le taux de reconnaissance, il suffit alors de compter le nombre de chiffre correctement reconnu par ligne, et on peut ainsi étudier l'influence du paramètre d et tester les limites du classifieur.

Le script «*computeProbabilitiesEuclidian.m* » effectue ce calcul.

## Analyse des résultats

Tous les résultats décrits dans cette section ont été générés avec le script « *benchmarkEuclidian.m* ».

Pour tester une configuration particulière (tester une seule valeur de d), exécuter « *main.m* ».

Le principe de ce test est de choisir plusieurs valeurs pour d, est d'afficher le taux de reconnaissance ainsi obtenu.

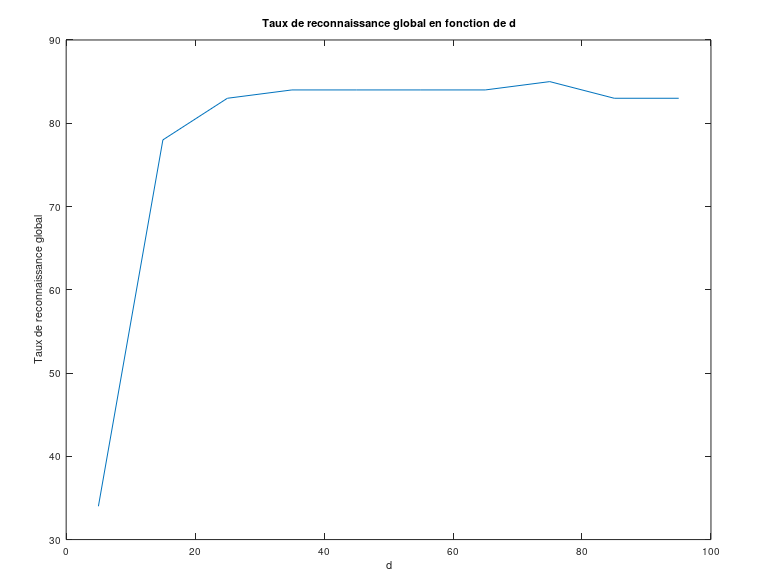
Intéressons-nous d'abord au taux de reconnaissance global.

Figure 3: Taux de reconnaissance global en fonction de d

On constate que la courbe est logarithmique et que puisque le nombre de lignes d est limité par le rectangle englobant, le taux de reconnaissance va plafonner à environ 83%.

Une autre caractéristique importante est qu'à partir de d = 15, le taux devient intéressant (78%).

On peut donc dire que d'une manière globale, pour ce classifieur on peut se contenter d'une valeur relativement faible pour d, et que si l'on souhaite améliorer le taux de reconnaissance, il faudra essayer d'autres méthodes.

Étudions à présent le taux de reconnaissance par chiffre, afin de vérifier si certains chiffres sortent de cette tendance (page suivante).

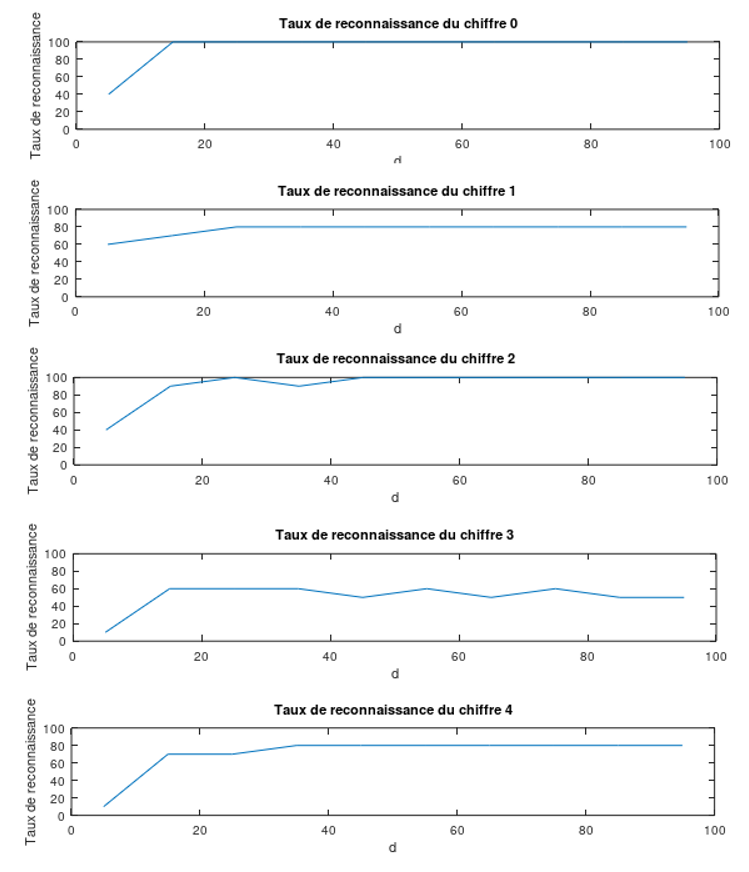


Figure 4: Taux de reconnaissance de 0-4

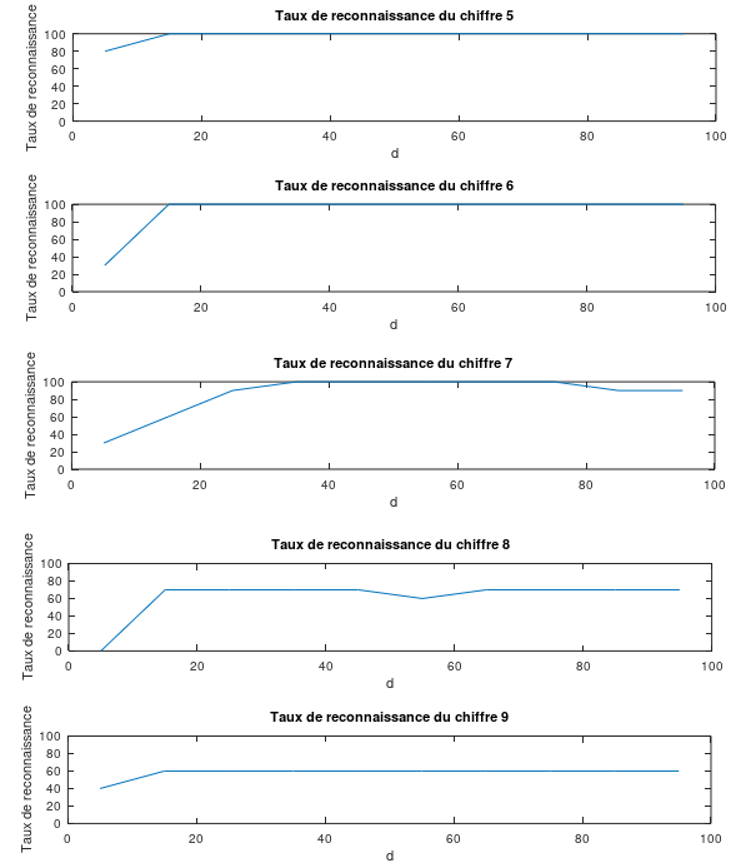


Figure 5: Taux de reconnaissance de 5-9

On remarque immédiatement que les chiffres 5, 0 et 6 sont ceux qui sont le mieux reconnus 100% à partir de d = 15.

Cela peut s'expliquer par le fait que les profils de ces chiffres sont assez distincts des autres chiffres, donc il y a peu de chances de se tromper.

On pourrait alors de demander pourquoi le 9 plafonne à 60%, alors que le 6 (9 à l'envers) atteint 100%.

La raison est que la personne ayant écrit les chiffres des images « *app.tif* » et « *test.tif »* écrit le 6 et le 9 d’une manière très différente :

* le rond du 9 est très arrondi tandis que celui du 6 est beaucoup plus aplati.
* la plupart des 6 ont un trait qui dépasse pour terminer l'arrondi, ce qui n'est pas le cas du 9.
* Le trait du 9 est assez vertical alors que celui du 6 suit une courbe assez prononcé.

On ne peut donc pas considérer ici que le 9 est un 6 à l'envers, donc il est normal que les taux de reconnaissances soient différents.

En revanche, cela n'explique pas les 60% du 9.

Ceci est dû au fait que certains 9 (5ème, 9ème et 10ème) sont légèrement étirés horizontalement avec une petite boucle légèrement décalé, ce qui du point de vue du profil du chiffre fait penser à un 7.

C'est d'ailleurs le résultat du classifieur pour d = 95 :

Une image contenant table

Description générée automatiquement

Le taux des chiffres 1, 4 et 8 suivent la courbe du taux de reconnaissance globale et plafonnent à 80% (70% pour le 8).

Enfin les chiffres 1, 2 et 7 sont plutôt bien reconnus même avec des valeurs de d relativement faibles, mais il faut un d un peu plus grand que pour 5, 0, et 6 pour obtenir des résultats intéressants.

En considérant les taux de reconnaissance par chiffre, on peut dire qu'il suffit de prendre une valeur autour de d = 30 pour obtenir des résultats satisfaisants, augmenter plus sa valeur n'apportant que très peu de différences.

L'analyse des taux par chiffres a permis de contraster la conclusion que l'on avait donnée en s'intéressant uniquement au taux de reconnaissance global:

d = 35 au lieu de d = 15.

Finalement, ce classifieur dont les performances sont rapidement atteintes et limitées aux alentours de 80%, peut-être paramétré avec une valeur pour d égale à environ 35. Pour d=35, on obtient les résultats suivants (lancer  le script « *main.m*» et choisir d=35) :

Une image contenant table

Description générée automatiquement

# **Classification par densités et KPPV**

## Principe et Calcul des densités

Le principe de ce classifieur est de découper le rectangle englobant un chiffre en n-zones verticales et m-zones horizontales et ainsi on obtient un vecteur à *d = n\*m* composantes. Ce classifieur se base sur l’algorithme d’apprentissage des K plus proche voisin (*K-NN: K-Nearest Neighbors*).

Pour chaque zone, pour obtenir la densité de pixels blancs, on compte le nombre de pixels blancs que l’on divise par le nombre total de pixels. Cette procédure est répétée pour chaque chiffre de l’image d’apprentissage.

Le script « *getDensities.m* » réalise cette opération.

## Apprentissage du classifieur

L’apprentissage consiste à appliquer la procédure de calcul des densités à chaque chiffre de l’image d’apprentissage. Le résultat issus de ces calculs sont ensuite conservé dans un fichier dénommé *densities.mat*.

Pour stocker les densités on utilise une matrice à 3 dimension dont la première correspond au nombre de chiffre de l’image et les deux dernières aux nombre de zones verticales et horizontales, elles servent à stocker les densités des chiffres. Pour pouvoir enregistré ces densités on transforme la matrice à 3 dimensions en 2 dimensions. On refait cette procédure pour analyser l’image test.

Le script « *trainKNN.m* » réalise cette opération.

## Décision

Pour déterminer les vecteurs de probabilité d’appartenance à chacune des classes, on applique l’algorithmes des K plus proches voisins. Ce algorithme consiste à déterminer les K voisins les plus proche du chiffre que l’on cherche à classer.

Pour applique cet algorithme on commence par créer une liste de voisins les plus proches du chiffrer qu’on cherche à classer, puis on déterminer le pourcentage que ce chiffre en question soit le chiffre 0, puis le 1 et ainsi de suite jusqu’au 9.

On obtient ainsi une liste de probabilités qui nous dit à combien de pourcent le chiffre qu’on cherche à classer appartient à chaque classe.

Le script « getKNN.m » réalise cette opération.

## Analyse des résultats

Le résultats décrits dans cette section ont été générés avec le script « *benchmarkKNN.m* ».

Comme pour le « *benchmarkEuclidian.m* », le principe de ce test est de choisir plusieurs valeurs pour m, n et k, est d'afficher le taux de reconnaissance ainsi obtenu.

Pour ce premier test, nous faisons varier le nombres de zone verticales est égale au nombre de zone horizontal et k.

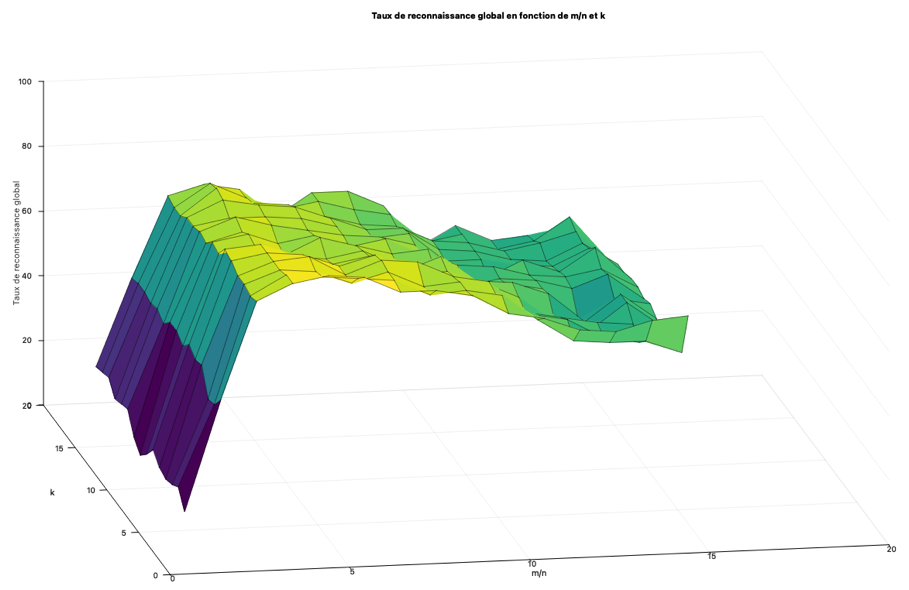


Figure 6: Tracé des taux de reconnaissance global en fonction de m/n et k

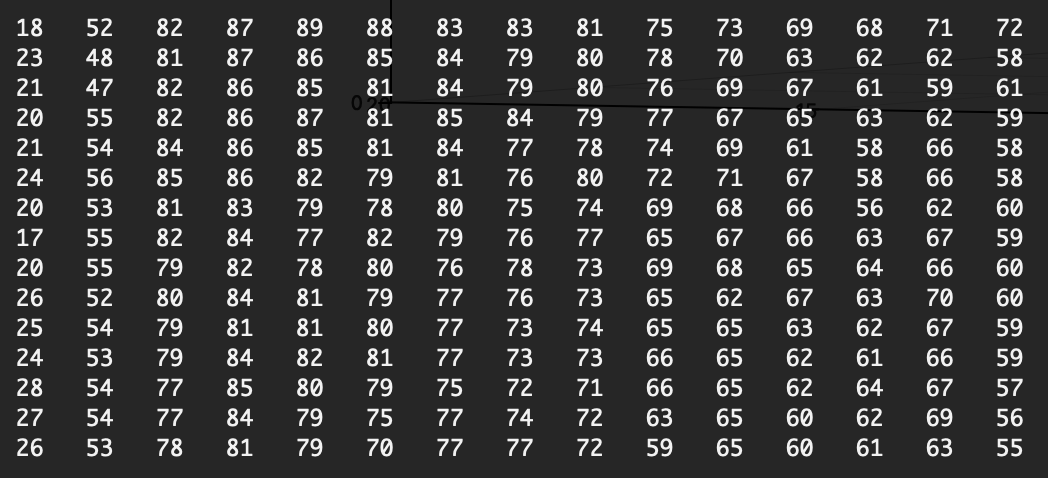


Figure 7: Taux de reconnaissance global en fonction de m/n et k (m/n en ligne et k en colonne)

Le tracé de la figure 6 nous montre l’évolution des taux de reconnaissances lorsqu’on fait varier m/n et k de 1 à 15 sachant que m est égale à n. On aperçoit sur ce tracé couplé avec la figure 7 qui montre la liste des taux de reconnaissance que dans un premier temps plus le couple m/n augmente plus la reconnaissance est meilleure et après m = n = 5, le taux de reconnaissance diminue.

Cette variation peut être expliqué par le fait que quand le nombre de zones (m/n) inférieur à 4, la densité de chaque zone représente une grande surface du chiffre donc on a plus d’incertitude sur la classe du nombre qu’on cherche à analyser, et quand le nombre de zones dépasse 6, la densité de chaque zone est précises, car la zone à analyser est trop petit ainsi le classifieur n’arrive plus à mettre la différence entre certains nombres à l’instar de 3 et 8 qui sont pour 9. Pour l’influence de la valeur de k, on retient que pour k égale à 1, le classifieur ne retient que le voisin le plus proche et pour k supérieur à 2, le classifieur n’arrive plus à distinguer la classe réelle du chiffre, car la classification donne un résultat contenant plusieurs classe.

Nous avons remarqué que plus la couleur tend vers le jaune, plus la reconnaissance est meilleur, et la meilleure classification est atteint pour m/n égale à 5 et pour k égale à 1. Le taux de reconnaissance pour ces paramètres est de 89%.

# **Combinaison de classifieurs**

A cette étape on réutilise les résultats des deux précédents classifieurs pour obtenir une meilleurs classification. Les méthodes de combinaison utilisés sont la somme et le produit.

En application les combinaison au meilleurs vecteur de probabilités des deux classifieurs, c’est-à-dire d égale à 75 (taux à 85%) pour le premier et m/n égale à 5 et k égale à 1 (taux à 89%), on obtient ainsi pour la somme un taux de 89% et pour le produit un taux de 89%.

# **Conclusion**