VISION PAR ORDINATEUR

TP2 : Textures

réalisé par LE Viet Man - promotion 15

FONCTIONNEMENT DU PROGRAMME

Pour faire ce TP, j'ai implémenté trois programmes : *GLCM* - calculer les matrices de co-occurrences et extraire un vecteur des caractéristiques pour chaque image, *Classification* - (comme un apprentissage supervisé) classifier des textures de test en utilisant l'algorithme k plus proche voisin, *SegTex* - identifier des textures dans une image.

Compilation des programmes

Pour compiler chaque programme, il faut modifier la variable INCLUDEDIR dans le fichier Makefile dans le répertoire du programme pour mettre le bon chemin pour le répertoire de OpenCV.

Ensuite, dans le répertoire du programme, vous tapez la commande `make clean`. Et puis, vous tapez la commande `make` pour compiler le programme.

Usage du programme GLCM

Avec ce programme, l'utilisateur peut calculer des vecteurs de caractéristiques pour un répertoire des images de texture. La sortie du programme est un fichier de texte qui stocke ces vecteurs de caractéristiques. Le code du programme s'est basé sur le code de Daniel Eaton (cvtexture.cpp). Le programme GLCM possède la commande suivante :

```
./glcm [options] <path>
```

où :

- `<path>` : le chemin du répertoire des images de texture
- `[options]` : le programme fournit 10 arguments :
  - `-h` : cet argument sert à afficher les informations de l'aide du programme
  - `-z` : si l'entrée est la base 2-RotInv_15_7_64, il faut choisir cette option
  - `-a, -b, -c, -d` : la distance est tour à tour 1, 2, 3, 4. La distance par défaut est 1.
  - `-w <int>` : le nombre de directions. Le programme ne supporte que quatre types de directions 4, 8, 12, et 16. Ces directions sont seulement convenables quelques distances précis, par exemple : la direction 8 ne supporte que la distance 1 et 3, la direction 12 ne supporte pas la distance 3 et la direction 16 ne supporte pas la distance 4. Le nombre de directions par défaut est 4.
  - `-g <int>` : le niveau de gris. Le programme supporte les niveaux de gris suivants : 8, 16, 32, 64, 128 et 256. Le niveau de gris par défaut est 8.

- `-p <int>` : des caractéristiques que l'utilisateur veut calculer. Le programme supporte 14 caractéristiques suivantes :

  0 - Contrast (CON)

  1 - Dissimilarity (DIS)

  2 - Homogeneity (HOM)

  3 - ASM

  4 - Energy

  5 - Maximum probabilite

  6 - Entropy (ENT)

  7 - GLCM Mean

  9 - GLCM Variance

  11 - Standard Deviation

  13 - GLCM Correlation

  Les formulaires sont trouvés sur http://www.fp.ucalgary.ca/mhallbey/equations.htm.

- `-s <path>` : le chemin de répertoire qui est utilisé pour enregistrer le fichier de résultat qui a l'extension ".glcm". Si l'utilisateur ne donne pas cette option, le fichier de résultat enregistrera sur le répertoire courant.

Usage du programme Classification

Le programme Classification classifie des textures dans un fichier d'entrée ".glcm" en utilisant l'algorithme k plus proche voisin. Le programme possède la commande suivante :

```
./classify [options] <filename>
```

où :

- `<filename>` : le fichier de test (.glcm)
- `[options]` : le programme fournit 5 arguments :
  - `-h` : cet argument sert à afficher les informations de l'aide du programme
  - `-k <int>` : la valuer k de l'algorithme k plus proche voisin. La valeur par défaut est 7.
  - `-d <int>` : type de calculer la distance. Le programme supporte trois types suivants :

    0 - NORMAL (la somme des différences des paires de deux caractéristiques)

    1 - COSIN

    2 - CORRELATION

    Les formulaires sont trouvés sur http://reference.wolfram.com/mathematica/guide/DistanceAndSimilarityMeasures.html.

- `-t <filename>` : le nom de fichier d'apprentissage (.glcm)
- `-s <path>` : le chemin du répertoire où le programme enregistrera le fichier de résultat. Si l'on ne donne pas cette option, le programme enregistrera l'image de résultat sur le répertoire courant. Le nom de l'image enregistré fini avec ".res".

Usage du programme SegTex

Le programme SegTex sert à identifier des textures dans une image. Le programme possède la commande suivante :

```
./segtex [options] <filename>
```

où :

- `<filename>` : le fichier d'image
- `[options]` : le programme fournit 6 arguments :
  - `-h` : cet argument sert à afficher les informations de l'aide du programme
  - `-k <int>` : la valeur k de l'algorithme k plus proche voisin. La valeur par défaut est 7.
  - -t <int> : la taille de la fenêtre de texture. Le programme supporte seulement des tailles 8, 16, 32, 64, 128, 256. La taille par défaut est 8.
  - -l <int> : le seuil de la distance. Le seuil par défaut est 0.001.
  - -b <filename> : la base d'apprentissage (.glcm)
  - -s <path> : le chemin du répertoire qui est utilisé pour enregistrer le fichier de résultat. Si l'utilisateur ne donne pas cette option, le fichier de résultat enregistrera sur le répertoire courant.

Quelques exemples

- Calculer un fichier des vecteurs de caractéristiques qui sont extrait aux images dans le répertoire `1-RotInv_13_6_training` avec deux distances 1 et 2, quatre directions et le niveau de gris par défaut 8. Pour chaque matrice de co-occurrence, on veut seulement extraire quatre caractéristiques (HOM, Energy, Entropy et Correlation). Le fichier est enregistré dans le répertoire `4parametre` :

  ```
  ./glcm -b -p 2 -p 4 -p 6 -p 13 -s 4parametre 1-RotInv_13_6_training
  ```

- Calculer des textures dans la base de test `1-RotInv_13_6_test.glcm` en utilisant la base d'apprentissage `1-RotInv_13_6_training.glcm` avec la valuer k est égale à 3, le type de distance est 1. Le fichier de résultat sera enregistré sur le fichier `1-RotInv_13_6_test_res.res` dans le répertoire courant :

  ```
  ./classify -k 3 -d 1 -t 1-RotInv_13_6_training.glcm 1-RotInv_13_6_test.glcm
  ```

- Identifier des textures dans l'image `mosaic_1.pgm` en utilisant l'algorithme k plus proche voisin avec k = 3, la taille de la fenêtre est 8x8 et le seuil pour la distance est 0.4, la base d'apprentissage est `1-RotInv_13_6_training.glcm`. Le fichier de résultat et l'image de résultat sont enregistrés sur le répertoire courant :

```
./segTex -k 3 -t 8 -l 0.4 -b 1-RotInv_13_6_training.glcm mosaic_1.pgm
```

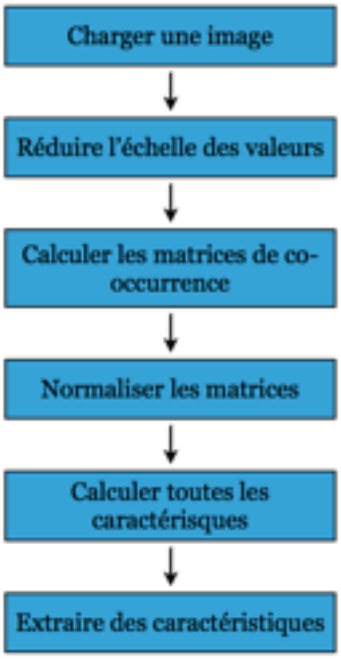
CLASSIFICATION DE TEXTURES

La première partie a pour but de faire de la classification de textures qui possède aussi deux phases : la phase d'apprentissage (calcul du vecteur de caractéristiques) - on doit extraire des vecteurs de caractéristiques dans des images de textures en calculant les matrices de co-occurrence; et la phase de classification - on utilise une stratégie de classification pour classifier une texture, on fait aussi l'évaluation sur des résultats.

Calcul du vecteur de caractéristiques

Dans cette étape, le programme doit calculer des matrices de co-occurrence pour extraire un vecteur de caractéristiques qui présente la texture dans une image. Avec les bases 1, 2, 3 et 4, chaque image est une texture. Donc, le programme traversera chaque image pour extraire des vecteurs et les stocker sur un fichier de résultat.

Le processus de ce programme comprend 6 étapes comme le diagramme à côté :



Processus de calcul d'un vecteur de caractéristique

- *Réduire l'échelle des valeurs* : le niveau de gris est une valeur très importante. Le calcul sur la matrice de co-occurrence avec un bas niveau de gris est plus moins qu'avec celui avec un haut niveau de gris, mais vous pouvez obtenir de mauvais résultats, car quand réduire l'échelle, on ne peut pas détecter des textures ou détecter mal. Mon programme soutient aux beaucoup de niveaux de gris (8, 16, 32, 64, 128 et 256), mais j'ai utilisé seulement le niveau de gris 8 dans mon test pour diminuer le temps de marcher du programme.
- *Calculer les matrices de co-occurrence* : j'ai corrigé le code de Daniel Eaton (cvtexture.cpp) parce que ce code applique un algorithme simple et efficace. On n'utilise que deux boucles imbriquées pour calculer n matrices (n est égale à la quantité des directions). La complexité de cet algorithme est seul $O(n^2)$.
- *Calculer toutes les caractéristiques* : En habituellement, on calcule seulement une caractéristique à la demande de l'utilisateur. Ce n'est pas efficace parce que le programme doit traverser encore une fois la matrice de co-occurrence pour chaque demande. Donc, dans mon programme, en corrigeant le code de Daniel Eaton, je n'ai utilisé que deux seules boucles imbriquées pour calculer toutes les caractéristiques. La performance de cette méthode a été approuvée par le temps de calcul tout de suite trois minutes sur toutes les quatre bases 1, 2, 3 et 4.
- *Extraire des caractéristiques* : l'utilisateur peut extraire seulement quelques caractéristiques pour les étapes suivantes.

- *Fichier de résultat (.glcm)* : pour faciliter la gestion, la comparaison des résultats et la comparaison de la convenance entre des bases de textures, le fichier de résultats du programme est structuré comme suivant :

  Cinq premières lignes sont cinq lignes d'informations (comme la tête d'un fichier) : la première est le chemin de la base de textures (base d'apprentissage ou base de test), la deuxième est le niveau de gris de la matrice de co-occurrence, la troisième et la quatrième est les distances et la quantité de directions de la matrice de co-occurrence, la dernière est des indices de caractéristiques.

  Ensuite, ce sont des lignes qui présentent des vecteurs de caractéristiques. Au début de chaque ligne est le nom de l'image de texture suivie de la classe de texture (est extrait au nom de l'image) est fin avec la vecteur de caractéristiques de cette image.

Base d'apprentissage et base de test

Dans cette partie, je dois diviser quatre bases en bases d'apprentissage et bases de test. Quatre premières bases 1, 2, 3, et 4 de texture ont pour but de tester ma stratégie d'apprentissage et de classification. Chaque base possède des caractères différentes. La base 1-RotInv_13_6 a 13 textures de Brodatz et chaque texture se présente sous 16 formes avec 6 angles de rotation (0, 30, 60, 90, 120, 150). La base 2-RotInv_15_7_64 a 15 textures de Brodatz, chaque texture se présente sous 16 formes avec 7 angles de rotation (0, 30, 60, 90, 120 et 150). La base 3-RotInv_16_10 a 16 textures de Brodatz avec 10 angles de rotation (0, 20, 30, 45, 60, 70, 90, 120, 135 et 150). La dernière base 4-Brodatz32 a 32 textures de Brodatz, chaque texture se présente sous 16 formes avec 4 angles de rotation (o, r, rs et s).

Avec les caractères ci-dessus, je veux tester le dernier angle de chaque forme de chaque texture et la dernière forme de chaque texture. Donc, les bases de test comprennent les images qui se présentent le dernier angle de chaque forme de chaque texture (avec les bases 1, 2 et 3, c'est l'angle 150; avec la base 4, c'est l'angle s) et les images qui sont la dernière forme de chaque texture (par exemple : avec la texture bark de la base 1, les images bark_###_15.pgm seront les textures de test. Cette façon de diviser garantit aussi le taux 1/4 entre les textures de test et les textures d'apprentissage.

Classification de textures

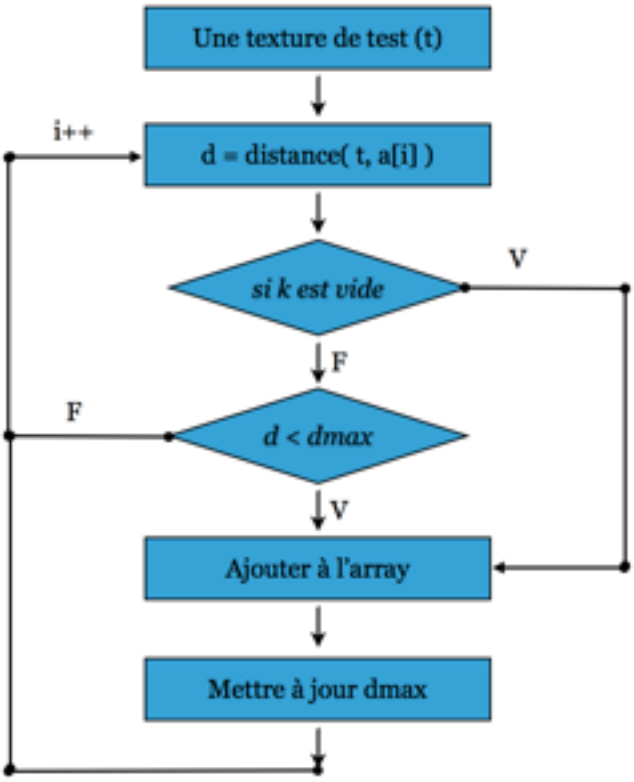
Il y a deux problèmes dans cette partie. Ce sont la stratégie de classification et la méthode de calculer la distance entre deux vecteurs de texture.

- **La stratégie de classification**

La stratégie de classification que j'ai utilisé est la méthode k plus proche voisin. C'est une méthode simple qui prend en compte des k textures d'apprentissage dont l'entrée est la plus proche de la texture de test, selon une distance à définir. La classe de la texture de test est la classe la plus représentée parmi les k textures d'apprentissage.

Traditionnellement, on calcule souvent toutes les distances entre la texture de test avec les textures d'apprentissage. Ensuite, on les range pour obtenir k textures la plus proche de la texture de test.

Même si la moins complexité de l'algorithme de ranger soit maintenant $O(n\log n)$, mais si la valeur n est plus grande que 10000, on doit attendre très longtemps. Cela influence sur la performance du programme. Donc, j'ai utilisé seulement un array de k éléments. Une texture d'apprentissage peut ajouter a l'array si la distance entre cette texture et la texture de test est plus moins que la plus haute distance (dmax) de k éléments dans cet array ou si cet array est encore vide. L'algorithme est comme le diagramme suivant.

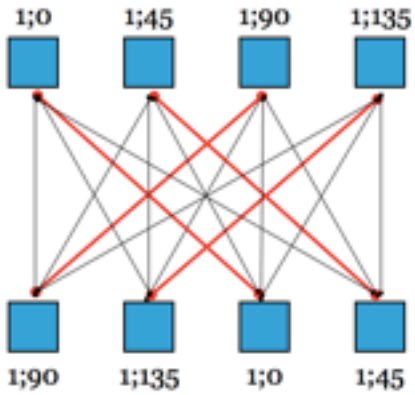


Mon algorithme k plus proche voisin
t : une vecteur de test, a : des vecteurs d'apprentissage, dmax : la plus haut distance de k élément dans cet array

Avec cet algorithme, le programme doit encore traverser sur toutes les textures d'apprentissage, mais puisque la valeur k soit plus moins que la quantité des textures d'apprentissage, le programme marche plus rapidement. En complexité, mon algorithme est $O(n^2)$, mais cet algorithme a moins de temps de calcul.

- **La distance entre deux textures**

Pour calculer la distance entre deux textures, j'ai appliqué la méthode l'appariement bipartie qui est un algorithme dans la théorie de graphe. On l'utilise toujours pour calculer la similarité entre deux graphes. Dans notre cas, on peut considérer le vecteur de texture comme un graphe où chaque noeud est une matrice de co-occurrence et des coordonnées du noeud sont des caractéristiques calculées sur cette matrice de co-occurrence. Ainsi la distance entre deux vecteurs de texture est égale à la distance d'appariement bipartie entre deux graphes des matrices de co-occurrence. Cette distance est calculée par la somme de distance des appariements où deux matrices de co-occurrence



Chaque noeud est une matrice de co-occurrence avec deux valeurs : la distance et la direction. La distance entre deux textures est la somme de distance des appariements (les arrêtes rouges)

peuvent apparier si la distance entre eux est la plus moins (voir la partie *la distance entre deux vecteurs* pour plus d'information).

On permute généralement les matrices de co-occurrence de texture d'apprentissage et calculer la distance de chaque permutation avec la texture de test. La plus moins distance est la distance entre la texture de test et une texture d'apprentissage. La permutation est une meilleure méthode, mais elle demande beaucoup de temps d'exécuter. Donc, j'ai simplifié cette méthode et je l'appelle la distance "naïve". Cette méthode fait une seule boucle qui traverse des matrices de co-occurrence dans la texture de test. Pour chaque matrice, elle choisit une matrice dans la texture d'apprentissage la plus proche (cette matrice dans la texture d'apprentissage ne sera pas choisie dans la prochaine fois). La distance "naïve" n'est pas optimale mais efficace sur le temps de calcul.

Dans mon test, j'ai testé toutes les deux méthodes permutation et "naïve" pour trouver la meilleure.

- **La distance entre deux vecteurs**

La distance entre deux vecteurs est juste la distance entre deux matrices de co-occurrence. Dans mon programme, j'ai implémenté trois types de distance entre deux vecteurs. Ce sont la distance Cosin, la distance Corrélation et la distance Normal - la somme des différences entre des paires de deux caractéristiques. J'ai aussi testé tous les trois types de distance pour trouver la meilleure distance.

SEGMENTATION DE TEXTURES COMPLEXES

Dans la seconde partie, j'ai combiné deux parties *Calcul de vecteur de caractéristique* et *Classification de textures* ci-dessus pour identifier des textures dans une image qui mélange de plusieurs textures.

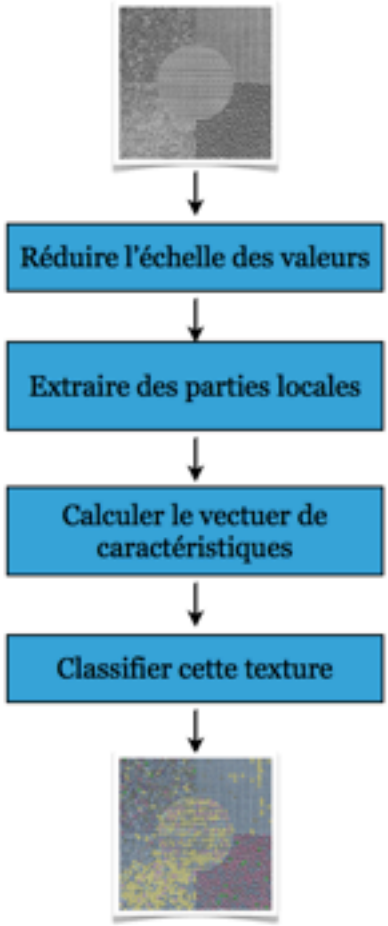
Bases d'images 5, 6, 7

Les bases d'images 5, 6 et 7 de l'Université de Oulu serons les bases de test dans cette partie. La base 5-Segmentation52 possède cinq mosaiques de textures, la base 6-SupervSegm12 possède 12 mosaiques de textures et la base 7-Composition_512_EQ possède 11 mixtures de textures qui sont composées d'orge et de riz. J'ai identifié des textures dans toutes les images de la base 5 et 7, mais j'ai identifié seulement des textures dans 7 premières mosaiques de la base 6. J'ai utilisé tour à tour les base 1, 2, 3 et 4 comme la base d'apprentissage.

Processus du programme

Le processus du programme comprend 4 étapes principales comme le diagramme à côté :

- *Extraire des parties locales* : pour chaque image, le



Le processus de la segmentation des textures

programme la divise en des parties avec une taille donnée par l'utilisateur. Le programme soutient des tailles de la fenêtre 8x8, 16x16, 32x32, 64x64, 128x128 et 256x256.

- *Calculer le vecteur de caractéristiques* : pour chaque partie divisée, le programme calcule des matrices de co-occurrence pour extraire un vecteur des caractéristiques. Les distances et les directions dépendent de ces valeurs de la base d'apprentissage.
- *Classifier cette texture* : après avoir calculé le vecteur de caractéristique, le programme fait la classification pour ce vecteur. Si la plus basse distance entre ce vecteur et des vecteurs d'apprentissage est moins que le seuil, cette partie est classifiée à une classe de texture. Sinon cette partie ne fait pas d'aucune classe. Des parties classifiées seront colorées son cadre par une couleur et des parties qui présentent une même texture, auront une même couleur.

ANALYSE DES RÉSULTATS

Tous les tests que j'ai fait, a utilisé les paramètres suivants : 8 niveaux de gris, deux distances (1, 2), quatre directions (0, 45, 90, 135).

Choix des caractéristiques

A partir de la matrice de co-occurrence, on peut calculer des caractéristiques. Chaque caractéristique joue des rôles différents et il y a des corrélations entre eux. Donc, choisir des caractéristiques est très important. Cela influence sur le résultat.

J'ai testé quatre types de combinaison des caractéristiques. Ce sont :

- *4 paramètres* : Homogénéité, Energie, Entropie et Corrélation dans lesquels l'Homogénéité est une valeur importante de groupe des caractéristiques représentant le contraste, l'Energie et l'Entropie sont les valeurs de groupe représentant la discipline et la Corrélation est la plus importante valeur de groupe statistique descriptive.
- *13 paramètres* : toutes les caractéristiques que le programme soutient sans l'Energie car l'Energie est calculé par ASM.
- *7 paramètres* : Homogénéité, ASM, Entropie, Moyen I et J, Variance I et J
- *6 paramètres* : Homogénéité, ASM, Entropie, Variance I et J

Les résultats de test 4 types de combinaisons sur 4 bases de textures sont suivants (j'ai utilisé la valeur k = 3) :

	4 paramètres	**13 paramètres**	**7 paramètres**	**6 paramètres**
Base 1	91.91%	93.75%	91.54%	90.44%
Base 2	86.36%	78.48%	76.67%	78.18%
Base 3	94.38%	94.91%	93.93%	93.93%
Base 4	89.97%	74.51%	71.71%	70.36%

A partir de ces résultats, je trouve que même si les résultats de 4 paramètres soit moins que celle de 13 paramètres, l'écart entre deux combinaisons n'est pas grand et surtout le nombre des paramètres est plus bas. Donc, selon moi, le premier type de combinaison (4 paramètres) est la meilleure combinaison.

La distance entre deux textures

Comme j'ai dit ci-dessus, la méthode de calculer la distance entre deux textures en utilisant la permutation exige beaucoup de temps de calcul. Ainsi que j'ai utilisé une autre méthode. C'est la méthode "naïve". Pourtant, une chose s'apparaît. Y a-t-il la différence dans le résultat de deux façons de calcul de la distance ? Est-ce que cela influence sur le résultat du programme ? Deux tables suivantes répondront aux problèmes.

	Permutation	**Naïve**
Normal	12	12
Cosin	0.51309	0.715381
Corrélation	1.43808	2.01963

Comparaison entre la distance "permutation" et la distance "naïve" quand appliquer des distances différentes de deux vecteurs.

	Permutation	**Naïve**
4 paramètres	93.01%	91.91%
13 paramètres	94.74%	93.75%

Comparaison la performance de la distance "permutation" et de la distance "naïve" avec des types de combinaison. Les tests avec k = 3, la distance Cosin, sur la base 1.

Alors, deux tables ci-dessus montre que le décalage entre deux méthodes n'est pas grand. Car la méthode "naïve" marche plus rapide, cette méthode est plus efficace. Donc, dans les tests suivants, j'ai utilisé seulement la méthode "naïve".

La distance entre deux vecteurs

La façon de calcul de la distance entre deux vecteurs (ou deux matrices) influence beaucoup sur le résultat du programme. Ainsi que choisir une façon de calcul de la distance entre deux vecteurs est très importante. Selon mon test, la distance Cosin est la meilleure. Cela est prouvé par la table à côté. Alors, j'utiliserai la distance Cosin pour les tests suivants.

	Normal	**Cosin**	**Corrélation**
Base 1	9.56%	93.75%	93.38%
Base 2	38.18%	79.09%	78.48%
Base 3	6.25%	93.84%	92.77%
Base 4	10.69%	60.86%	60.20%

Comparaison les méthodes de calcul de la distance entre deux vecteurs avec k = 7, la méthode "naïve" et 13 paramètres

K plus proche voisin

La méthode k plus proche voisin est une méthode simple et facile à implémenter. Pourtant, son efficace dépend de la valeur k. Choisir la valeur k fera changer le résultat de classification. On peut utiliser la méthode validation croisée (cross-validation, en anglais) pour choisir une meilleure valeur

k, mais c'est très complexe pour ce TP. Donc, j'ai fait un test avec les valeurs k différentes pour choisir la meilleure k de chaque base de textures. Le résultat de mon test est suivant :

	k	4 paramètres	13 paramètres	7 paramètres	6 paramètres
Base 1	**1**	93.38%	94.49%	94.49%	91.91%
	3	91.91%	93.75%	91.54%	90.44%
	5	91.91%	93.01%	91.91%	87.50%
	7	90.44%	93.75%	92.28%	87.87%
	9	88.97%	93.38%	91.91%	86.03%
	11	88.60%	91.18%	90.81%	83.82%
	13	85.29%	90.81%	90.81%	82.72%
Base 2	**1**	87.27%	77.88%	77.88%	78.48%
	3	86.36%	78.48%	76.67%	78.18%
	5	86.06%	78.79%	78.48%	77.88%
	7	86.06%	79.09%	78.18%	78.18%
	9	84.85%	77.88%	78.48%	77.58%
	11	85.15%	78.79%	78.48%	77.58%
	13	85.15%	78.48%	78.48%	78.18%
Base 3	**1**	85.80%	91.34%	92.23%	86.88%
	3	89.02%	93.13%	93.57%	92.05%
	5	91.61%	94.11%	93.48%	92.41%
	7	93.13%	93.84%	93.84%	93.66%
	9	94.02%	94.64%	93.84%	94.46%
	11	94.38%	94.91%	93.93%	93.93%
	13	94.64%	94.55%	93.84%	94.38%
Base 4	**1**	95.23%	81.58%	79.93%	73.36%
	3	89.97%	74.51%	71.71%	70.72%
	5	84.87%	64.31%	61.68%	64.14%
	7	81.74%	60.86%	58.39%	60.69%
	9	77.96%	57.40%	55.43%	58.06%
	11	77.14%	56.09%	54.93%	56.74%
	13	75.33%	52.30%	52.30%	54.93%

A partir de cette table, je trouve que avec des bases où la quantité des échantillons est basse, la petite valeur k obtient le meilleur résultat. A l'inverse, des bases où la quantité des échantillons est grande, la grande valeur k obtiendra le meilleur résultat.

De plus, je peut choisir la valeur k et le type de combinaison des caractéristiques correspondant à chaque base de textures. En particulier, ma choix est comme suivante :

- Base 1 : k = 3, 13 paramètres
- Base 2 : k = 3, 4 paramètres
- Base 3 : k = 11, 13 paramètres
- Base 4 : k = 3, 4 paramètres

Classification des textures

Avec la base 1, avec k = 3 et la vecteur de textures comprend 13 caractéristiques, le résultat est 93.75%. La matrice de confusion de la base 1 est suivante :

	bark	brick	bubbles	grass	leather	pigskin	raffia	sand	straw	water	weave	wood	wool	
bark	20		1		1									**22**
brick		20				1								**21**
bubbles	1	1	20											**22**
grass				20										**20**
leather				1	20				1					**22**
pigskin						20	5	1					1	**27**
raffia							13							**13**
sand							1	20						**21**
straw									19					**19**
water										19				**19**
weave									1		21			**22**
wood										2		21		**23**
wool							2						20	**22**
	21	**21**	**21**	**21**	**21**	**21**	**21**	**21**	**21**	**21**	**21**	**21**	**21**	

	bark	brick	bubbles	grass	leather	pigskin	raffia	sand	straw	water	weave	wood	wool
Précision	95.2	95.2	95.2	95.2	95.2	95.2	61.9	95.2	90.5	90.5	100.0	100.0	95.2
Rappel	90.9	95.2	90.9	100.0	90.9	74.1	100.0	95.2	100.0	100.0	95.5	91.3	90.9

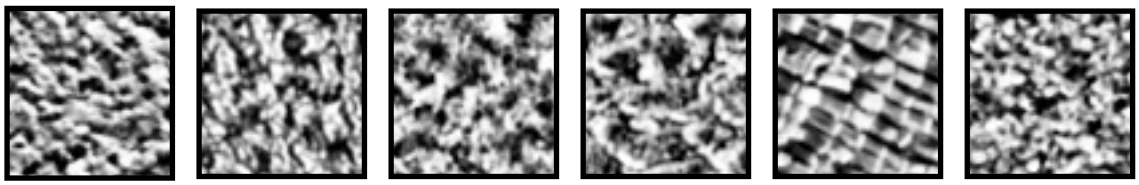
Le résultat de classification est bon. La plupart des classes sont bien séparées, mais deux classe raffia et pigskin sont mal séparées. La classe raffia a la basse valeur précision 61.9% et les fautes sont la plupart de la dernière forme de la texture raffia. Cela se passe parce que peut-être cette forme n'est pas appris. La classe pigskin a aussi la base valeur rappel 74.1%, mais la majorité des fautes se représente dans la classe raffia.

La base 2 a la plus bas résultat. Dans la base 2, il y a plusieurs classes mal séparées comme d004, d019, d029, d084 et d092.

	d004	d009	d016	d019	d021	d024	d029	d032	d053	d057	d068	d077	d084	d092	d093	
d004	9					1										**10**
d009		22				1										**23**
d016			22													**22**
d019				17									1	9		**27**
d021					22											**22**
d024	11					20				4						**35**
d029							17						4			**21**
d032							5	22								**27**
d053									22							**22**
d057	2									18			2			**22**
d068											22					**22**
d077												22				**22**
d084													15			**15**
d092				5										13		**18**
d093															22	**22**
	22	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	

	d004	d009	d016	d019	d021	d024	d029	d032	d053	d057	d068	d077	d084	d092	d093
Précision	40.9	100.0	100.0	77.3	100.0	90.9	77.3	100.0	100.0	81.8	100.0	100.0	68.2	59.1	100.0
Rappel	90.0	95.7	100.0	63.0	100.0	57.1	81.0	81.5	100.0	81.8	100.0	100.0	100.0	72.2	100.0

Sa cause principale est le ressemblance des textures dans six classes.



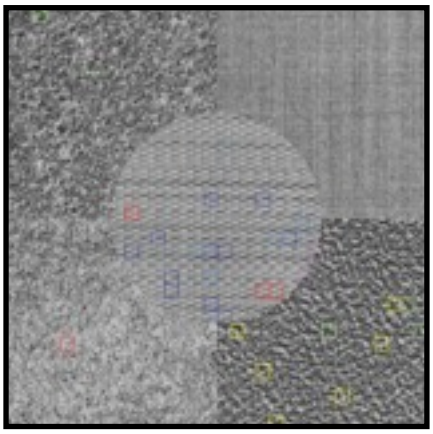
d004 *d024* *d019* *d092* *d084* *d029*

Ces classes causera des maux résultats dans l'identification des textures.

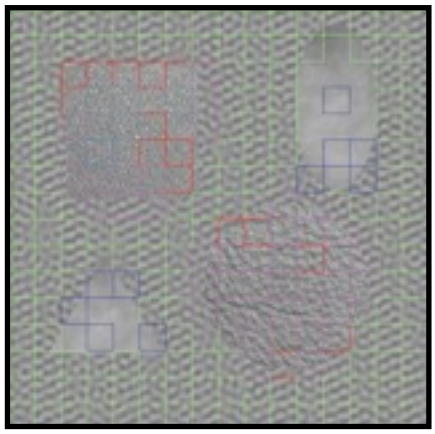
Segmentation des textures

Dans cette partie, j'ai détecté des textures dans les images dans la base 5, 6, 7. Quelques résultats suivants :

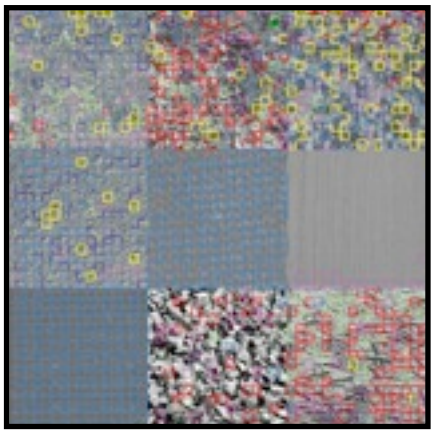
- *Base d'apprentissage - base 1 et base de test - base 5*



mosaic_1 avec la taille de la fenêtre 16, le seuil 0.1
8 textures : leather, straw, sand, wool, weave, raffia, pigskin, grass.

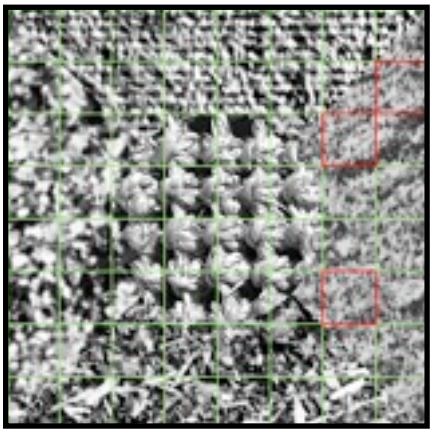


mosaic_3 avec la taille de la fenêtre 32, le seuil 0.05
6 textures : water, wool, pigskin, wood, weave, sand.

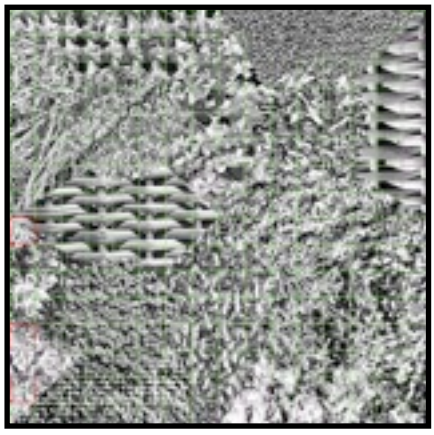


mosaic_5 avec la taille de la fenêtre 8, le seuil 0.1
13 textures : weave, straw, wood, pigskin, wool, water, sand, bark, grass, leather, brick, bubbles, raffia.

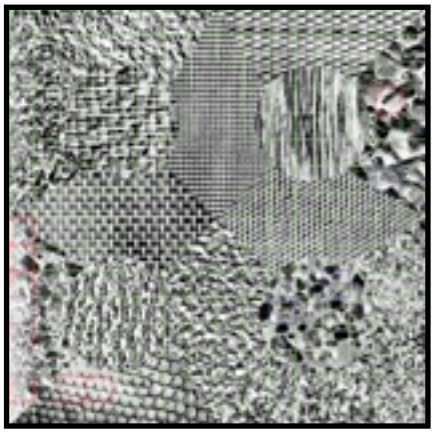
- *Base d'apprentissage - base 1 et base de test - base 6*



mosaic_4 avec la taille de la fenêtre 32, le seuil 0.4
2 textures : grass, straw



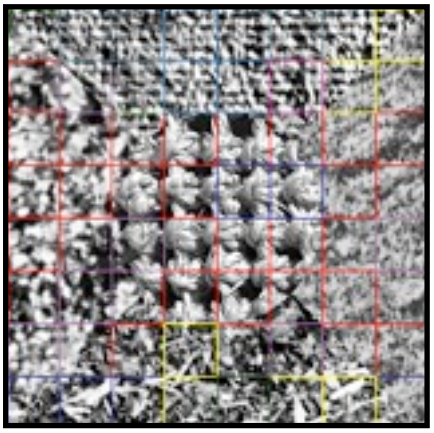
mosaic_7 avec la taille de la fenêtre 32, le seuil 0.4
3 textures : grass, straw, bark



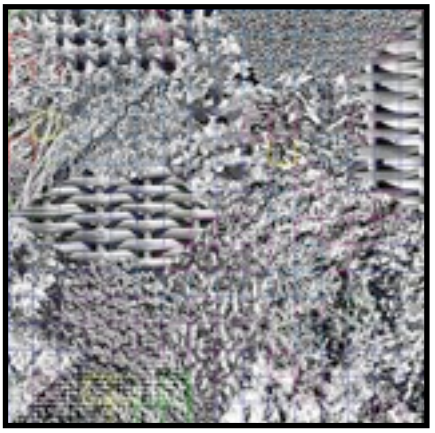
mosaic_6 avec la taille de la fenêtre 32, le seuil 0.4
4 textures : grass, straw, bark, brick

Le résultat ici est très mauvais parce que le seuil est trop grand.

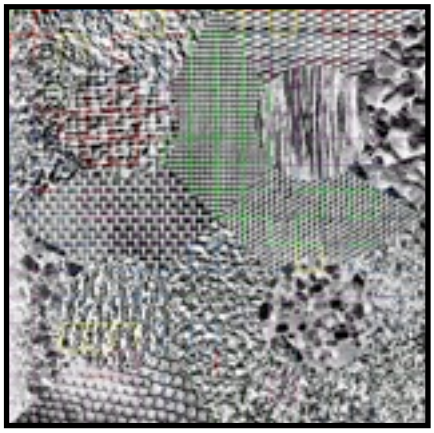
- *Base d'apprentissage - base 2 et base de test - base 6*



mosaic_4 avec la taille de la fenêtre 32, le seuil 0.4
9 textures : d029, d019, d084, d092, d024, d004, d032, d009, d057



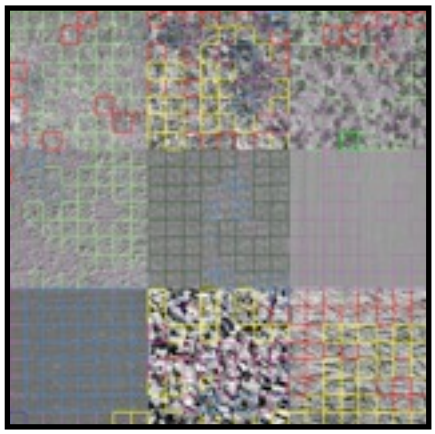
mosaic_7 avec la taille de la fenêtre 32, le seuil 0.4
13 textures : d029, d093, d009, d019,d084, d016, d024, d068, d092, d021, d032, d004, d057



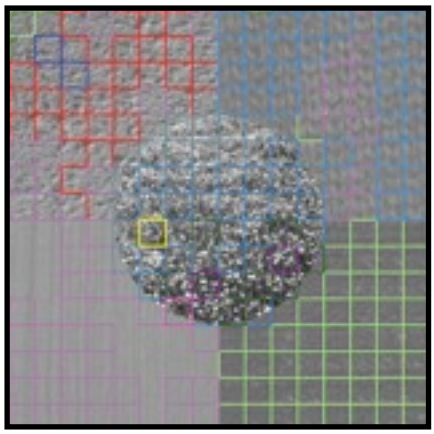
mosaic_6 avec la taille de la fenêtre 32, le seuil 0.4
14 textures : d029, d084, d019, d092, d032, d024, d004, d009, d093, d068, d057, d077, d016, d021

Des classes mal classifiées apparaissent beaucoup dans les résultats.

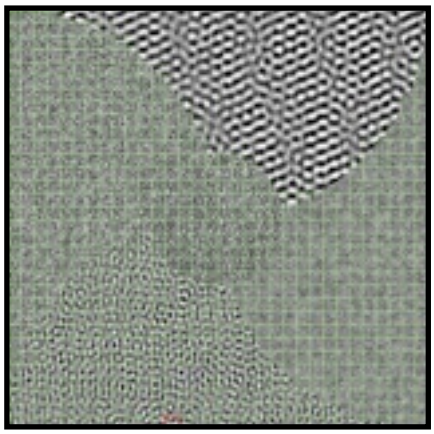
- *Base d'apprentissage - base 3 et base de test - base 5*



mosaic_5 avec la taille de la fenêtre 16, le seuil 0.4
12 textures : Sand, Paper, Rattan, Cloth, Weave, Straw, Canvas, Wood, Raffia, Grass, Leather, Wool



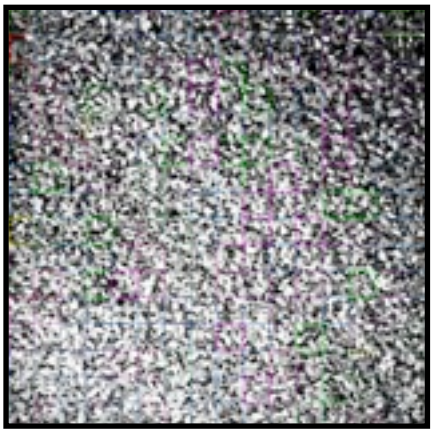
mosaic_4 avec la taille de la fenêtre 16, le seuil 0.4
11 textures : Straw, Sand, Rattan, Weave, Paper, Canvas, Grass, Leather, Raffia, Cloth, Wood,



mosaic_2 avec la taille de la fenêtre 16, le seuil 0.4
2 textures : Canvas, Cloth

Le programme ne peut pas détecter des textures dans la haute partie de l'image mosaic_2.

- *Base d'apprentissage - base 4 et base de test - base 7*

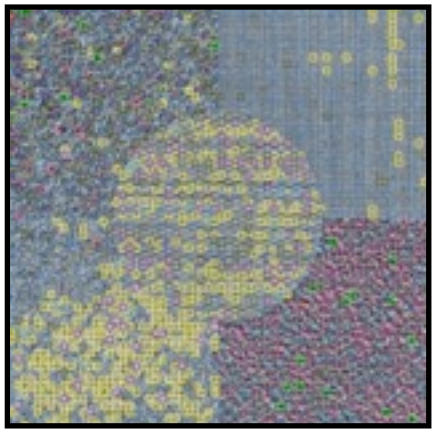


b000r100 avec la taille de la fenêtre 32, le seuil 0.1
17 textures : d51, raffia2, straw2, d5, d95, seafan, burlap, paper, d10, tree, d11, reptile, d4, peb54, d52, ice, ricepaper

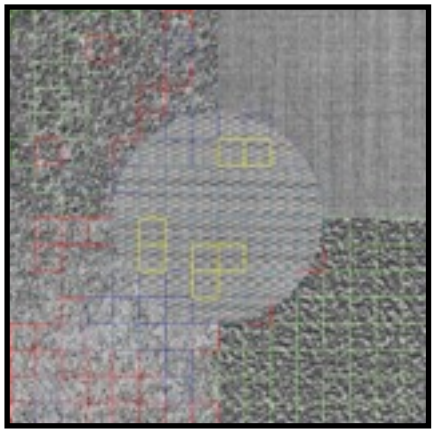
b020r080 avec la taille de la fenêtre 16, le seuil 0.4
19 textures : d51, d95, d5, d10, seafan, d4, straw2, ricepaper, paper, burlap, d52, d11, reptile, tree, ice, peb54, image15, image17, raffia2

b080r020 avec la taille de la fenêtre 32, le seuil 0.4
14 textures : d10, d52, raffia2, d4, paper, seafan, reptile, d95, ice, burlap, ricepaper, peb54, image15, tree

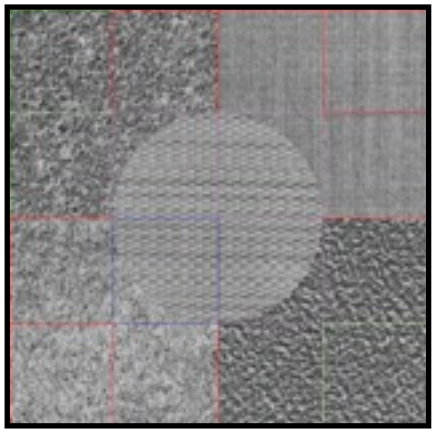
- *Les résultats avec des tailles différentes*



la taille 8

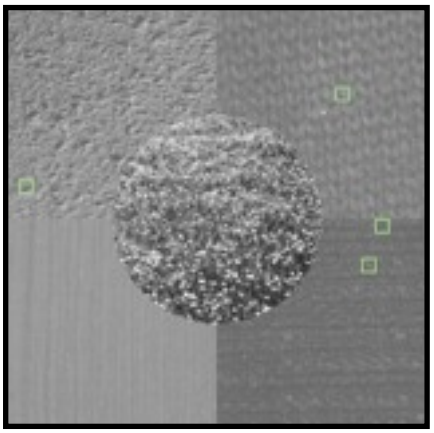


la taille 32

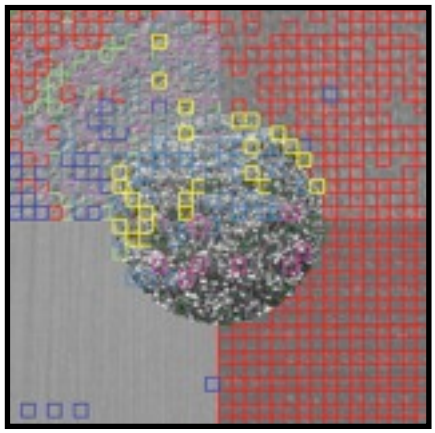


la taille 128

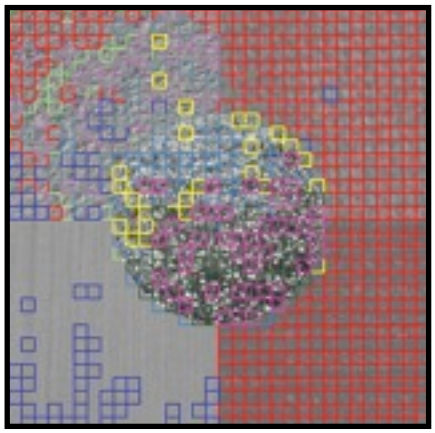
- *Les résultats avec des seuils différents*



le seuil 0.01



le seuil 0.1



le seuil 0.5

CONCLUSION

La texture est un élément important dans une image et contribue beaucoup dans plusieurs domaines, tels que la segmentation, la recherche d'image et le synthèse d'image. La matrice de co-occurrence est une simple méthode mais elle exige beaucoup de temps de calcul. Le petite niveau de gris peut diminuer le temps de calcul de la matrice, mais cela peut cause la manque des textures ou des fautes. L'appariement bipartie est une bonne méthode de calcul de la distance entre deux textures, mais la permutation est un faible. On peut diminuer la quantité des textures calculée en utilisant un rayon autour de texture de test.