Documentation Outil Interne Ptrans

Introduction Distances DTC

Le programme Distance DTC est le programme qui nous a servis pour :

- Le calcul de distance des descripteur
- Le tri des resultats
- Le calculs d'un classement prenant à la fois le score de la distance couleur et texture
- La récupération et comparaison du classement avec la classification "manuelle"
- Le formatage d'un csv pour les résultats de l'étape précédente

Explication des classes

Descripteur CLD / Descripteur HTD :

Ces classes servent à extraire les descripteurs de chaque image contenue dans les fichiers textes. Ces deux classes possèdent des attributs et méthodes très similaire.

Un descripteur HTD possèdent une suite de nombre mis à la suite sur UNE SEULE ligne.
Un descripteur CLD possèdent une suite de nombre mis à la suite sur TROIS lignes.

Chaque ligne correspond respectivement aux informations Y CR et CB à l'intérieur de l'image.

HTD possède donc un seul vecteur 'int pour stocker les données tandis que CLD en possède trois.

Ces deux classes possèdent toutes deux les méthodes suivantes

```
void pushBack(int a (Pour HTD ) / int a , int Y (Pour CLD ));
double calculDistance(DescripteurHTD/CLD b);
```

PushBack permet de rajouter un nombre à l'intérieur du/des vecteurs d'int. Le deuxième int pour l'implémentation CLD correspond au numéro du vecteur dans lequel on rajoute l'int.

CalculDistance permet de récupérer la distance entre le descripteur courant et un autre du même type.

Distance :

Cette classe a le seul objectif de contenir la donnée de distance entre deux images. Elle possède deux attributs, la valeur de la distance ainsi que le numéro d'identification de l'image comparée

Un seul identificateur est stocké car cette classe fonctionne uniquement dans un cadre de vecteur à deux dimensions

```
std::vector<std::vector<Distance>> data;
```

Les indexs du premier index font alors office de numéro d'identification

Si nous reprenons notre exemple précédent

```
data[0]  ;
```

correspond alors à la liste des distances entre l'image 0 et le reste de la collection.

La classe distance implémente aussi l'opérateur de comparaison "<". Cela nous permet d'appliquer la fonction sort sur un vecteur de Distance

NB lorsque nous calculons la distances pour deux descripteurs nous appliquons la méthode suivante.
Soit deux descripteurs

A B C E D F G H
A' B' C' E' D' F' G' H'

Distance = $(A-A)^2 + (B-B')^2 + (C-C')^2 + (D-D')^2 + (E-E')^2$

RankPerImg :

RankPerImg est le conteneur des données de distance.
Comme mentionné précédemment ces données sont stockées dans un double vecteur de Distance.
Les méthodes les plus importantes de cette classe sont

*void outputrankHybrid(RankPerImg CLD, RankPerImg HTD, int nbimg, std::vector<std::vector<Distance> * > &container,double ratio);*

Il s'agit de la méthode la plus importante de la classe. C'est elle qui est capable créer un classement des descripteur par rapport à leur distances respectives sur l'aspect de la couleur et de la texture.

- CLD est le rankPerImg contenant les informations sur les distances de couleurs. HTD sur celles des textures.

- NbImg est le nombre d'images total de la collection, 7

- std::vector<std::vector<Distance> * > &container est la variable qui contiendra le résultat du traitement.

- Ratio est la variable qui indique quelle est l'importance que l'on veut mettre sur chacun des descripteurs. Cette variable fonctionne avec une échelle allant de 0 à 10. Si on ne veut que des distances de couleur dans notre classement il suffit de mettre 0 si on ne veut que l'information des textures 10 devra être mis à l'intérieur. SI on veut que les deux informations soient traités de manière égale on met 5.

SortRank();

Permet de trier les distances par ordre croissant.
Attention néanmoins outputrankHybrid() part du principe que toutes les distances ont été rentrée dans le même ordre dans ses arguments CLD et HTD
SI sortRank() est utilisé avant outputrankHybrid() cela fausserait complètement le jeu de données

normalisation();

Étant donné que nous utilisons plusieurs type de descripteur sur des images très différentes les unes des autres il est possible que l'ordre de grandeur des distance ne soit pas le même. Cette fonction permet donc de mettre sur le même rapport toutes les distance du RankPerImg. La normalisation consiste juste à centrer et réduire les données à l'intérieur de chaque Vecteur de Distance

SubRankCmp

SubRankCmp est la dernière classe importante de notre programme.
C'est elle qui a pour rôle d'extraire de stocker et de traiter la classification d'image faite "à la main".
Une classification est un fichier texte suivant le format suivant

IdImageOrigine IdImageRessemblante IdImageRessemblante IdImageRessemblante

En voici un example

```
0 26 33 32 59 51 69 25 70 37 61
1 2 114 111 3 4 112 113 12 11 77 79
44 43 45 192 190 191 39 40 46 47 174
52 53 195 193 194 54 55 196 50 51
86 80 84 83 107 122 92 95 96 10 123
141 142 143 176 177 197 172 209 200 199
144 139 140 156 157 158 165 166 167 164
169 170 168 171 172 201 202 203 204 173
180 178 179 184 185 186 192 174 181 176
207 208 209 210 206 205 200 198 197 142
71 20 31 58 63 72 101 103 7 29 68
2 5 9 13 14 15 17 34 38 57 67
3 10 16 28 56 88 90 101 112 115
4 0 11 12 15 17 32 33 45 62 89
7 3 10 13 14 20 21 29 31 58 68
8 2 3 13 14 16 21 23 31 34 38
9 10 14 17 22 38 46 49 56 62 72
10 7 8 13 14 15 46 56 68 73 88
71 7 10 20 31 56 58 63 68 101 103
33 0 26 32 37 50 51 59 61 69 70
63 7 20 31 58 68 103 108 71 101 106
65 8 21 23 35 60 67 98 34 38 72
102 26 69 100 107 116 25 32 62 89 122
106 7 135 16 18 20 31 58 68 101 103 105
```

Sur la première ligne nous pouvons voir que les images ressemblant le plus à 0 sont : 26, 33, 32 …..

Les méthodes importantes de SubRankCmp sont

```
void extractSubRank(std::string nameFile);
```

Permet d'extraire la classification se trouvant dans le fichier ayant pour nom nameFile.

```
double compareReversed(std::ofstream & myfile, std::string outputFile, const RankPerImg & rank, int nbImg );
```

Cette fonction écrit dans un stream ofstream le résultat de la comparaison entre la classification humaine et celle trouvée par le programme.

La comparaison marche de la manière suivante.

On prend chaque ligne de la comparaison humaine et on extrait son équivalent ordinateur. Nous extrayons ensuite les n plus courtes distances trouvées par l'ordinateur pour l'image d'origine. On regarde ensuite combien font partie de la classification humaine et un pourcentage est fait à partir de cette stat.

- Rank est la variable contenant la classification faite par l'ordinateur

- Int nbImg est le nombre d'image que nous voulons comparer entre les deux classification la limite étant le nombre d'image dans la classification

NB : Dans le cadre d'utilisation de notre programme un grand nombre d'appel est fait sur cette fonction. Pour différencier les résultats entre eux nous écrivons d'abord le nom ou l'identification de la comparaison. Cette dernière est stockée dans outputFile

NBB : le format des données de sortie est en CSV.

Documentation Programme MEPG

Le code de génération des descripteurs utilise MPEG7 et OpenCV2.4.13.

-Pour la partie descripteur, on a utilisé la réalisation sur github:
https://github.com/cdmh/MPEG7FexLib

-Pour configurer OpenCV 2.4.13

Télécharger le code:

http://opencv.org/releases.htm

1. Utiliser CMAKE pour gérer les codes source et produire la solution Visual Studio 2015(Ou d'autre edition)

2. Compiler le code source et obtenir dynamic-link library(fichier .dll) et statically-linked library(fichier .lib). Attention, pour obtenir l'édition debug et release, il faut compiler le code source en mode debug et en mode release

3. Si vous utilisez en IDE Visual Studio, configurez le Property Manager, mettez ...opencv\build\include\opencv et ...opencv\build\include\opencv2 à VC++

   Directories->Include Directories. Mettre le chemin de statically-linked library à VC++ Directories->Library Directories. Ajouter tous les noms de statically-linked library à Linker->additional dependencies

La module pour faire entrer des images et sauvegarder le résultat est au dessous.

Classe	Fonction/Macro/Attribut	Utilisation	Fichier
	getPapierPeint()	Parser de la lecture des images. Dans “list.txt” on indique le chemin d’images ainsi que le nombre total d’images, finir par “***”. exemple : F:\images 200 ***	main.cpp
	#define DES_CSD 1 #define DES_SCD 2 #define DES_CLD 3 #define DES_DCD 4 #define DES_EHD 5 #define DES_HTD 6 #define TYPE_DES 3	Indiquer le type de descripteur qu’on va calculer. TYPE_DES sert dans la fonction main: stock.calculerDescriptor(TYPE_DES);	MacroTypeDES.h
PapierStock	std::map<std::string,std::vector<PapierPeint*>> stock;	Une banque de papier peint, la clé de map (attribut stock) est le nom de dossier qui contient des images de papier peint.	PapierStock.h
PapierStock	void ecrireDescriptor()	La fonction pour écrire les descripteurs dans le fichier. Ici on a aussi utilisé le macro pour différents descripteurs.	PapierStock.cpp
PapierPeint	void calculDescriptor(int type)	Calculer le descripteur qui est indiqué par type	PapierPeint.cpp
PapierPeint	cv::Mat image	Utiliser la classe Mat de OpenCV pour stocker et gérer l’image	PapierPeint.h