Le traitement d'images



Détection d'objets par le classificateur de Haar





La méthode de Haar : détection d'objet

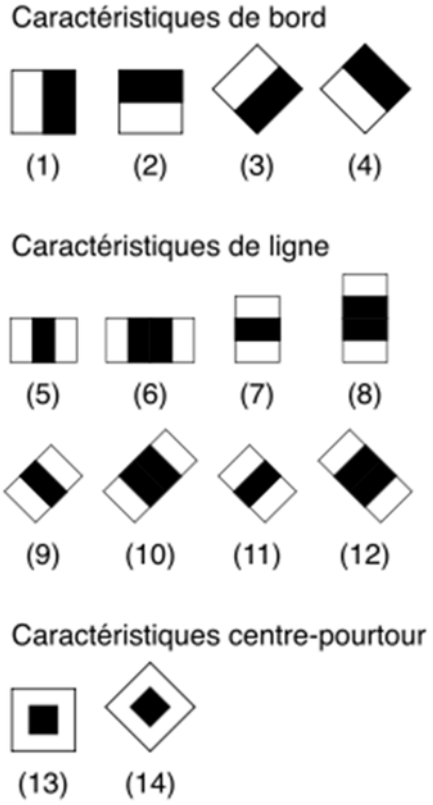
Les chercheurs Paul Viola et Michael Jones en 2001 ont proposés une méthode de détection d'objet dans une image numérique. Elle fait partie des toutes premières méthodes capables de détecter efficacement et en temps réel des objets dans une image. Inventée à l'origine pour détecter des visages, elle peut également être utilisée pour détecter d'autres types d'objets comme des voitures ou des avions. La méthode de Viola et Jones est l'une des méthodes les plus connues et les plus utilisées, en particulier pour la détection de visages et la détection de personnes. Une étape préliminaire et très importante est l'apprentissage du classificateur. Il s'agit d'entraîner le classificateur afin de le sensibiliser à ce que l'on veut détecter. Pour cela, il est mis dans deux situations.
La première où une énorme quantité d'images positives (avec l'objet à détecter) lui sont présentés et la deuxième où, à l'inverse, une énorme quantité de d'images négatives lui sont présentés.
Concrètement, une banque d'images contenant des objets à détecter est passée en revue afin d'entraîner le classificateur. Ensuite, une banque d'images ne contenant pas d'objets est passée.

1) L'apprentissage du classificateur

1.1) Caractéristiques d'une image

a) Extraction des caractéristiques d'une image

Viola et Jones proposent d'utiliser des caractéristiques, c'est à dire une représentation synthétique et informative, calculée à partir des valeurs des pixels. Viola et Jones définissent des caractéristiques très simples, les caractéristiques pseudo-Haar, qui sont calculées par la différence des sommes de pixels de deux ou plusieurs zones rectangulaires adjacentes.



La figure ci-dessus donne des exemples des caractéristiques proposées par Viola et Jones à 2, 3 ou 4 rectangles, dans lesquelles la somme des luminances de pixels sombres est soustraite de la somme des luminances des pixels blancs.

Ces caractéristiques ont été choisies de façon à détecter des motifs. Par exemple, la reconnaissance des visages est rendue possible par:

- La variation de l'intensité de la lumière entre les yeux et le nez (caractéristique n°2)
- la variation de l'intensité de la lumière entre les yeux et les pommettes (caractéristique n°3)

Les caractéristiques sont calculées en soustrayant la somme des pixels noirs à la somme des pixels blancs.

b) image intégrale

Pour calculer rapidement et efficacement ces caractéristiques sur une image, les auteurs proposent également une nouvelle méthode, qu'ils appellent « image intégrale ». C'est une représentation sous la forme d'une image, de même taille que l'image d'origine, qui en chacun de ses points contient la somme des luminances des pixels situés au-dessus de lui et à sa gauche. Plus formellement, l'image intégrale ii au point (x,y) est définie à partir de l'image i par :

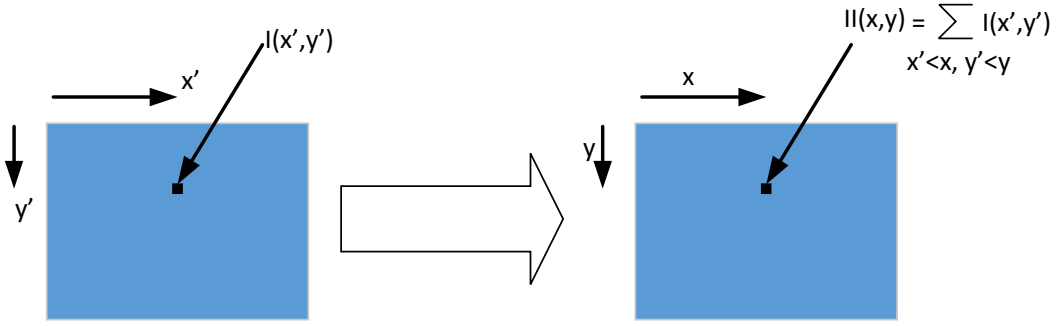
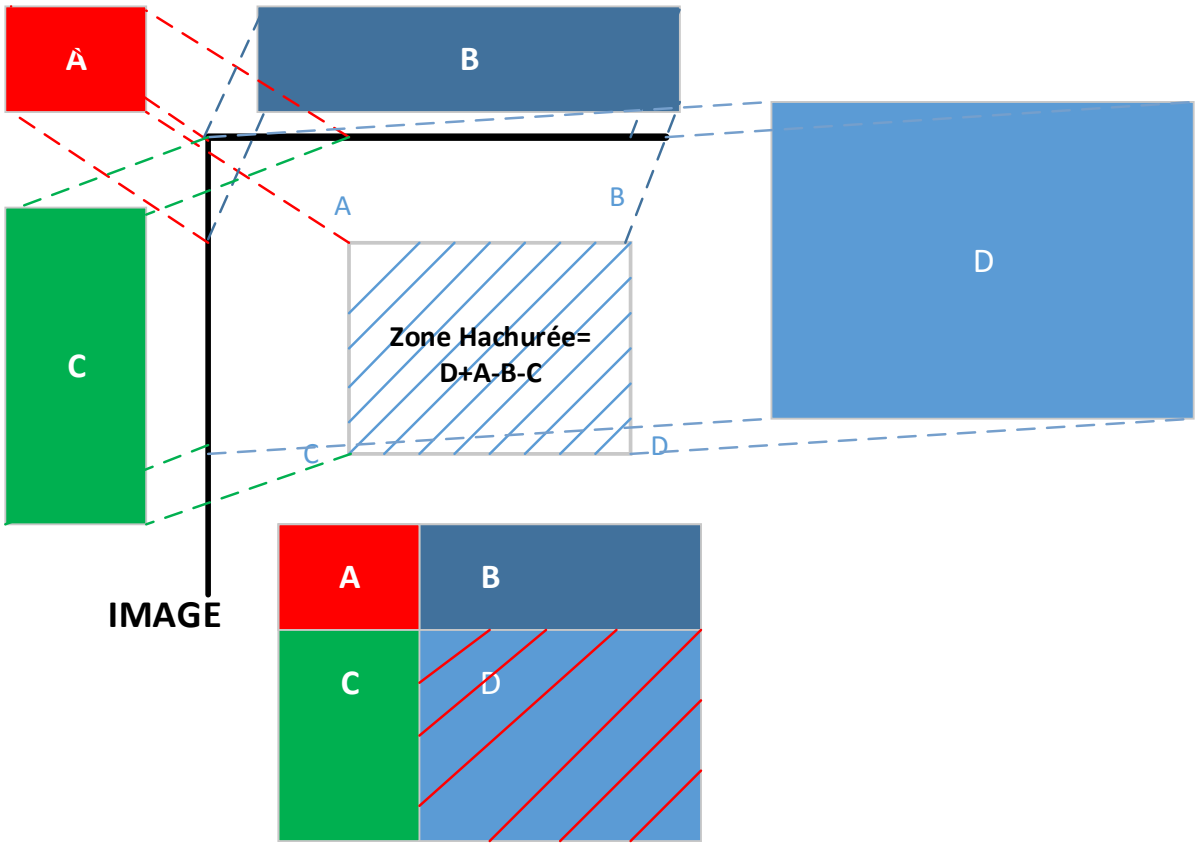


Image i Image ii: image intégrale

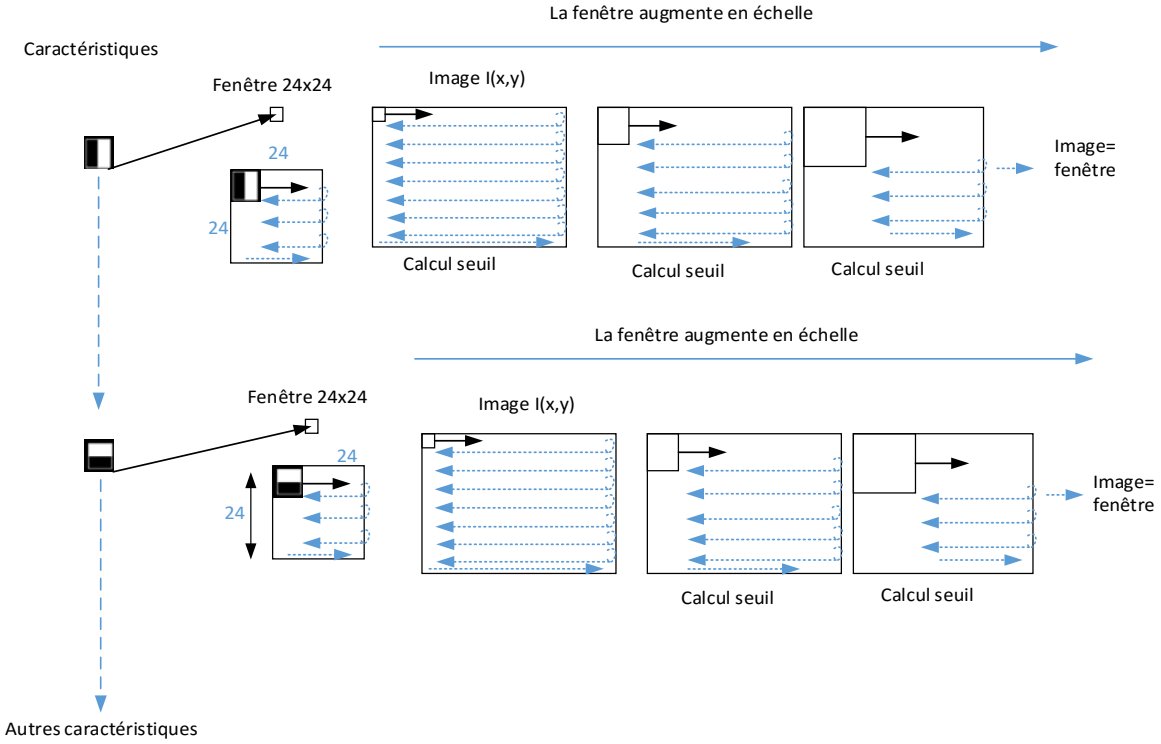
Illustrons le calcul de la somme de la luminance des pixels dans la zone hachurée ABCD ci-dessous



Seulement 4 accès à l'image intégrale nous permet de calculer la somme de la luminance de la zone ABCD (D+A-B-C)
Une caractéristique pseudo-Haar formée de deux zones rectangulaires peut être calculée en seulement 6 accès à l'image intégrale, et donc en un temps constant quelle que soit la taille de la caractéristique.

c) Calcul

Les caractéristiques sont calculées à toutes les positions et à toutes les échelles dans une fenêtre de détection de petite taille, typiquement de 24 × 24 pixels ou de 20 × 15 pixels. Un très grand nombre de caractéristiques par fenêtre est ainsi généré, Viola et Jones donnant l'exemple d'une fenêtre de taille 24 × 24 qui génère environ 160 000 caractéristiques pour une image.



La taille de la caractéristique de pseudo-Haar est variable et permet d'augmenter le nombre de caractéristiques évaluées
En phase de détection, l'ensemble de l'image est parcouru en déplaçant la fenêtre de détection (24x24) d'un certain pas dans le sens horizontal et vertical (ce pas valant 1 pixel dans l'algorithme original). Les changements d'échelles se font en modifiant successivement la taille de la fenêtre de détection. Viola et Jones utilisent un facteur multiplicatif de 1,25, jusqu'à ce que la fenêtre couvre la totalité de l'image. Finalement, et afin d'être plus robuste aux variations d'illumination, les fenêtres sont normalisées par la variance des pixels de la fenêtre.

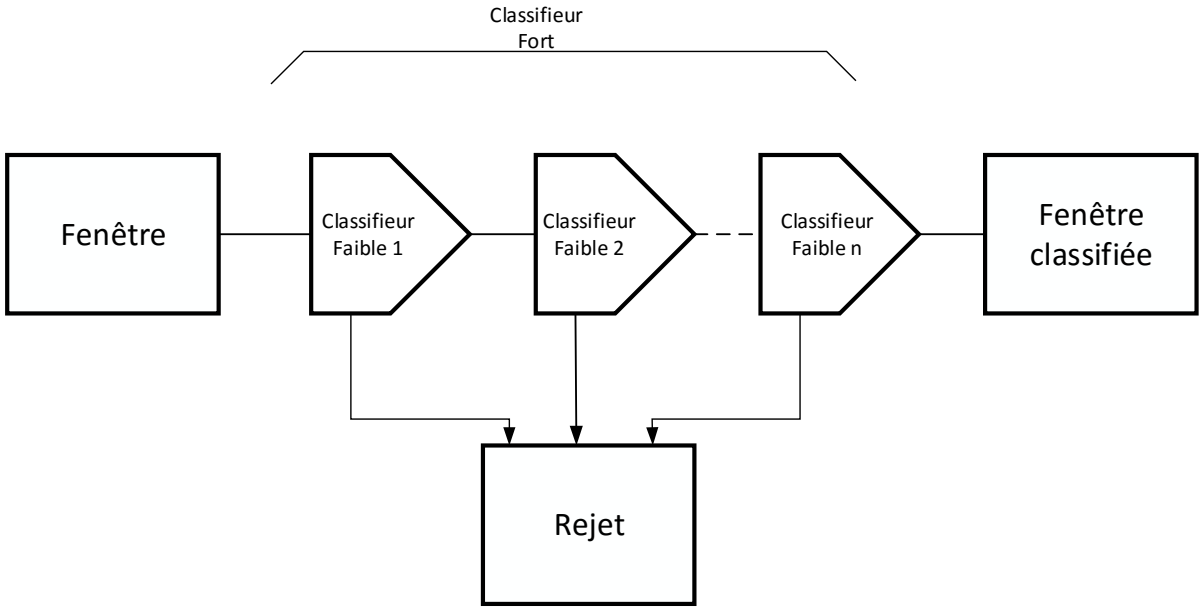
d) Elaboration du classificateur

Sélection de caractéristiques par boosting
Le deuxième élément clé de la méthode de Viola et Jones est l'utilisation d'une méthode de boosting afin de sélectionner les meilleures caractéristiques (sur les 14 disponibles). Le boosting est un principe qui consiste à construire un classificateur « fort » à partir d'une combinaison pondérée de classificateur « faibles », c'est-à-dire donnant en moyenne une réponse meilleure qu'un tirage aléatoire. Viola et Jones adaptent ce principe **en assimilant une caractéristique** à un classificateur faible, en construisant un classificateur faible qui n'utilise qu'une seule caractéristique.

Après avoir calculé toutes les caractéristiques provenant de chacune des formes de la fonction de Haar, c'est le temps de définir un seuil selon lequel on décide si l'image en traitement est un visage ou non. Chaque paramètre passe sur

toutes les images intégrales calculées à partir des images originales de la base de données adoptée. Ainsi, ces seuils seront enregistrés dans un fichier qui contiendra les poids de tous les classificateurs de Haar.
Une fois le calcul de la fonction Haar a été complété pour les images positives, la moyenne, l'écart-type, le min et le max sont calculés pour toutes les valeurs données par chaque image.
Ces paramètres sont calculés afin de catégoriser un classificateur comme bon ou mauvais par une distribution gaussienne : on commence à partir de la valeur moyenne et on élargit selon l'écart type, plus l'écart-type est grand, plus la reconnaissance sera pour les visages.
On doit faire le même calcul pour les images sans objets.
Pour tester les classificateur obtenus et décider quels sont les forts classificateur qu'on va utiliser ensuite dans la chaîne en cascade, chaque classificateur, ayant une distribution gaussienne, doit vérifier 3 conditions :
- Obtenir une reconnaissance d'objets de près de 100% (presque toutes les images d'objets à détecter sont détectées).
- Obtenir une reconnaissance sans objets très faible (la plupart des images sans objets ne sont pas détectées).
- Erreur totale pour le classificateur inférieure à 50%.

Si et seulement si ces trois conditions sont réunies, on continue avec le classificateur en traitement.



L'apprentissage du classificateur faible consiste alors à trouver la valeur du seuil de la caractéristique qui permet de mieux séparer les exemples positifs (image dans l'objet est présent) des exemples négatifs (image dont l'objet est absent).
Le classificateur se réduit alors à un couple (caractéristique, seuil). Au final, un classificateur est une association entre une caractéristiques pseudoHaar et un seuil. C'est ce qu'on appelle un classificateur faible.
Un classifieur h(x), composé d'un descripteur *f*, d'un seuil θ, et d'une parité *p*, donne une prédiction sur la classe à qui appartient *x* (ici, 1 pour visage et 0 pour non visage).

$$h(x, f, p, \theta) = \begin{cases} 1 \; si \; p.f(x) < p.\theta \\ 0 \; dans \; les \; autres \; cas \end{cases}$$

Un classificateur faible ne suffit pas à classifier une image mais c'est l'association de classificateur faible qui va permettre d'obtenir un classificateur fort.
Le nombre de classificateur étant trop élevé pour le calcul lors de la détection, l'algorithme de boosting va permettre de réduire le nombre de classificateur faible.

L'algorithme de boosting utilisé est en pratique une version modifiée d' AdaBoost, qui est utilisée à la fois pour la sélection et pour l'apprentissage d'un classificateur « fort ». Les classificateurs faibles utilisés sont souvent des arbres de décision. Un cas remarquable, fréquemment rencontré, est celui de l'arbre de profondeur 1, qui réduit l'opération de classification à un simple seuillage.
L'algorithme est de type itératif, à nombre d'itérations déterminé. À chaque itération, l'algorithme sélectionne une caractéristique, qui sera ajoutée à la liste des caractéristiques sélectionnées aux itérations précédentes, et le tout va contribuer à la construction du classificateur fort final. Cette sélection se fait en entraînant un classificateur faible pour toutes les caractéristiques et en élisant celle de ces dernières qui génère l'erreur la plus faible sur tout l'ensemble d'apprentissage. L'algorithme tient également à jour une distribution de probabilité sur l'ensemble d'apprentissage, réévaluée à chaque itération en fonction des résultats de classification. En particulier, plus de poids est attribué aux exemples difficiles à classer, c'est à dire ceux dont l'erreur est élevée.
Le classificateur « fort » final construit par AdaBoost est composé de la somme pondérée des classificateurs sélectionnés. Les classifieurs de précision plus élevée (taux d'erreur entre 0.1 et 0.3) sont sélectionnés au début de l'apprentissage, et les classifieurs moins précis (taux d'erreur entre 0.4 et 0.5) sont sélectionnés dans les dernières itérations.
L'apprentissage est réalisé sur un très large ensemble d'images positives (c'est-à-dire contenant l'objet) et négatives (ne contenant pas l'objet). Plusieurs milliers d'exemples sont en général nécessaires.
Après la phase d'apprentissage du classificateur, les différents seuils pour chaque classificateur faible sont enregistrés dans un fichier de métadonnées xml. Ce fichier sera exploité lors de la phase de détection.

2) La Détection

La détection s'applique sur une image de test, dans laquelle on souhaite déceler la présence et la localisation d'un objet. En voici les étapes :

- parcours de l'ensemble de l'image à toutes les positions, avec une fenêtre de taille 24 × 24 pixels et variable, application de la cascade à chaque sous-fenêtre, en commençant par le premier étage :
- calcul des caractéristiques pseudo-Haar de l'image utilisée par le classificateur de l'étage courant,
- calcul de la réponse du classificateur,
- passage ensuite à l'étage supérieur si la réponse est positive, à la sous-fenêtre suivante sinon, et enfin l'exemple est déclaré positif si tous les étages répondent positivement ;
- fusion des détections multiples : l'objet peut en effet générer plusieurs détections, à différentes positions et échelles ; cette dernière étape fusionne les détections qui se chevauchent pour ne retourner qu'un seul résultat.

3) La détection sur opencv en utilisant le format Iplimage

Pour commencer, nous allons mettre en place les premières briques du programme. On commence par inclure les fichiers nécessaires au programme.

```
#include <opencv/cv.h>
#include <opencv/highgui.h>
#include <stdio.h>
```

Ensuite on va déclarer deux de manière globale deux variables, le premier est le classificateur et le second une zone mémoire utilisée comme « buffer » pour la détection des visages. Le fait de les déclarer de manière globale n'est pas forcément idéal, mais le but est ici d'avoir le code le plus simple possible. Pour une application plus importante, il est recommandé de déclarer ces deux variables dans la partie nécessitant la détection de visage.

```
CvHaarClassifierCascade *cascade;
CvMemStorage *storage;
int key;
```

On rentre maintenant dans la fonction main, et nous allons déclarer tous les objets nécessaires par la suite. Nous avons besoin de peu d'élément, le premier sera une image (IplImage), le second sera le périphérique d'entrée de la caméra (CvCapture) et le dernier sera un int nous permettant de sortir de la boucle de traitement. Ce qui nous donne:

```
CvCapture *capture; IplImage *frame; int key;
```

Il faut ensuite initialiser tout ce beau monde. On commence par ouvrir le fichier du classificateur Haar

```
cascade = ( CvHaarClassifierCascade* )cvLoad(
« haarcascade_frontalface_alt.xml », 0, 0, 0 );
```

On ouvre le flux caméra avec ***cvCreateCameraCapture***

```
capture = cvCreateCameraCapture(CV_CAP_ANY)
```

Enfin on initialise l'espace mémoire

```
storage = cvCreateMemStorage( 0 );
```

Afin d'afficher tout cela, nous aurons besoin d'une fenêtre

```
cvNamedWindow( « Window-FT », 1 );
```

Nous sommes en C++, il ne faut donc pas oublier de libérer la mémoire avant la fin du programme, on doit alors effacer tous les objets précédemment créé.

```
cvReleaseCapture( &capture );
cvDestroyWindow( « Window-FT » );
cvReleaseHaarClassifierCascade( &cascade );
cvReleaseMemStorage( &storage );
```

Nous avons donc ici la base du programme. Si vous compilez vous ne devriez pas avoir grand-chose qui se passe et c'est normal, il manque l'essentiel, la boucle de traitement d'image ainsi que la fonction permettant l'affichage du tracking facial. C'est ce que nous allons voir dès maintenant.

La boucle de traitement

Cette étape est plutôt simple, mais elle est très importante, le principe est d'effectuer, à chaque image envoyée par la caméra au programme, la détection de visage. On en profite pour rajouter une petite commande permettant de quitter la boucle (et donc le programme) en appuyant sur la touche « q ». Voici la boucle en question

```
while( key != 'q' )
{
img= cvQueryFrame( capture ); detectFaces( img ); key = cvWaitKey( 10 );
}
```

Pas grand-chose à expliquer ici, une boucle while classique, on déclare une frame (image) qui correspondra à chaque instant à l'image envoyée par la caméra. La fonction ***detectFaces*** est la fonction que nous allons étudier juste après qui permet de faire la détection de visage à proprement parler. Si vous voulez tester votre caméra, il vous suffit de remplacer ***detectFaces(img);*** par ***cvShowImage(« Window-FT », img);***si vous compilez, vous devriez voir en temps réel ce que votre caméra est en train de filmer.

La fonction detectFaces

Il ne reste plus qu'à créer la fonction detectFaces afin de permettre le face tracking. Voici la fonction

```
void detectFaces( IplImage *img ) {
int i;
CvSeq *faces = cvHaarDetectObjects(img, cascade, storage, 1.1, 3 , 0,  cvSize( 40, 40 ) );
for( i = 0 ; i < ( faces ? faces->total : 0 ) ; i++ )
{
CvRect *r = ( CvRect* )cvGetSeqElem( faces, i );
cvRectangle( img, cvPoint( r->x, r->y ), cvPoint( r->x + r->width, r->y + r->height ), CV_RGB( 255, 0, 0 ), 1, 8, 0 );
}
cvShowImage( « Window-FT », img );
}
public static IntPtr cvHaarDetectObjects(
        IntPtr image,
        IntPtr cascade,
        IntPtr storage,
        double scaleFactor,
        int minNeighbors,
        int flags,
        MCvSize minSize
)
```

La fonction récupère en paramètre l'image de la caméra récupérée pendant la boucle de traitement. C'est sur cette image que toutes les opérations vont être effectuées par la suite. C'est là où vous allez apprécier OpenCV, car pour la détection de

visage, la méthode de Viola et Jones est déjà implémentée et deviens très facile à utiliser. Tout réside dans la fonction ***cvHaarDetectObjects***.

Le résultat de cette fonction est une série d'objets qui ont passé les critères de sélection définis par votre classificateur. On définit donc une CvSeq qui correspond à une séquence d'objet d'un même type, dans notre cas ce sera nos différents visages détectés.

Au niveau des paramètres de cette fonction ***img*** représente l'image a traité***, cascade*** est le classificateur choisi pour faire le test***, storage*** est l'espace mémoire nécessaire pour effectuer l'opération***, 1.1*** représente le « scale factor » l'augmentation de la fenètre (24x24) pour chaque analyse ***,*** minNeighbors=***3*** représente le nombre de rectangle voisins qui seront pris en compte pour la détection de l'image ***,*** plus ce nombre sera grand moins le nombre d'objet détecté sera grand mais la détection sera d'une très grande fiabilité

0 est un paramètre supplémentaire qui permet de rajouter des filtres particuliers par exemple un filtre de canny avec CV_HAAR_DO_CANNY_PRUNING*,* ***cvSize(40, 40)*** représente la taille minimale d'un objet dans la vidéo.

Nous avons maintenant l'ensemble des visages qui ont été détectés dans l'image dans notre CvSeq, et afin d'afficher le résultat à l'écran nous allons simplement dessiner un carré autour du visage détecté. Ensuite on rentre dans une boucle for qui va passer en revue tous les visages détectés, et pour chacun créer un rectangle autour du visage repéré.

Je ne détaille pas le dessin du rectangle, Enfin la dernière étape est d'afficher à chaque appel de fonction l'image de la caméra avec les carrés permettant de détecter les visages, grâce à la fonction ***cvShowImage(« Window-FT », img);***

Pour cette fonction le « ***Window-FT »*** est le nom de la fenêtre que vous avez créée et le ***img*** représente l'image à afficher dans la fenêtre.

4) Détection de Haar en utilisant l'objet Mat

On peut détecter un objet par la méthode de Viola Jones en utilisant l'objet Mat.
Il nous faut déclarer un classificateur de la classe CascadeClassifier et le charger

CascadeClassifier::CascadeClassifier(const string& filename) // méthode constructeur charge un classificateur

CascadeClassifier face_cascade;

face_cascade.load("C:/OpenCV243/data/Haarcascades/haarcascade_frontalface_alt2.xml");

void CascadeClassifier::**detectMultiScale**(const Mat& image, vector<Rect>& objects, double scaleFactor=1.1, int minNeighbors=3, int flags=0, Size minSize=Size(), Size maxSize=Size())

// Détecte les objets de différentes tailles dans l'image.
// Les objets détectés sont retournés comme une liste de rectangle

Les paramètres de la fonction ::**detectMultiScale**

- **cascade** – Classificateur fichier xml
- **image** – Objet du type Mat format CV_8U contenant l'image avec les objets détectés
- **objects** – Vecteur d'objet rectangle ou chaque rectangle contient l'objet détecté sur l'image.
- **scaleFactor** voir chapitre précédent.
- **minNeighbors** voir chapitre precedent
- **flags** – voir chapitre précédent.
- **minSize** – dimension minimum possible pour l'objet.
- **maxSize** – imension maximum possible pour l'objet

Exemple d'utilisation

```
#include "opencv2/objdetect/objdetect.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"

#include <iostream>
#include <stdio.h>

using namespace std;
using namespace cv;

int main( )
{
   Mat image;
   image = imread("lena.jpg", CV_LOAD_IMAGE_COLOR);
   namedWindow( "window1", 1 );   imshow( "window1", image );

   // Load Face cascade (.xml file)
```

```
    CascadeClassifier face_cascade;
    face_cascade.load( "C:/OpenCV243/data/Haarcascades/haarcascade_frontalface_alt2.xml" );

    // Detect faces
    std::vector<Rect> faces;
    face_cascade.detectMultiScale( image, faces, 1.1, 2, 0|CV_HAAR_SCALE_IMAGE, Size(30, 30) );

    // Draw circles on the detected faces
    for( int i = 0; i < faces.size(); i++ )
    {
       Point center( faces[i].x + faces[i].width*0.5, faces[i].y + faces[i].height*0.5 );
       ellipse( image, center, Size( faces[i].width*0.5, faces[i].height*0.5), 0, 0, 360, Scalar( 255, 0, 255 ), 4, 8, 0 );
    }

    imshow( "Detected Face", image );

    waitKey(0);
    return 0;
}
```