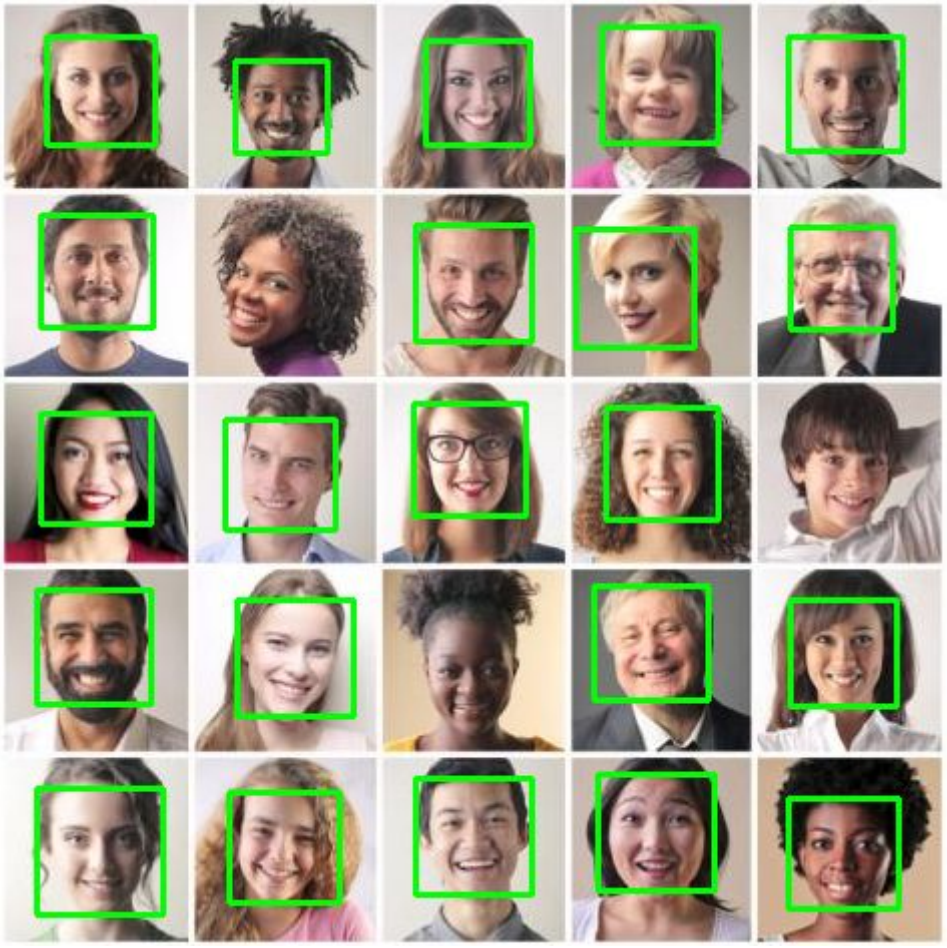
Antonin Bousquier
Ibrahim Elouard

CPE Lyon - 5 ETI - Robotique

15/11/2018

TP VISION : Cascade de Haar

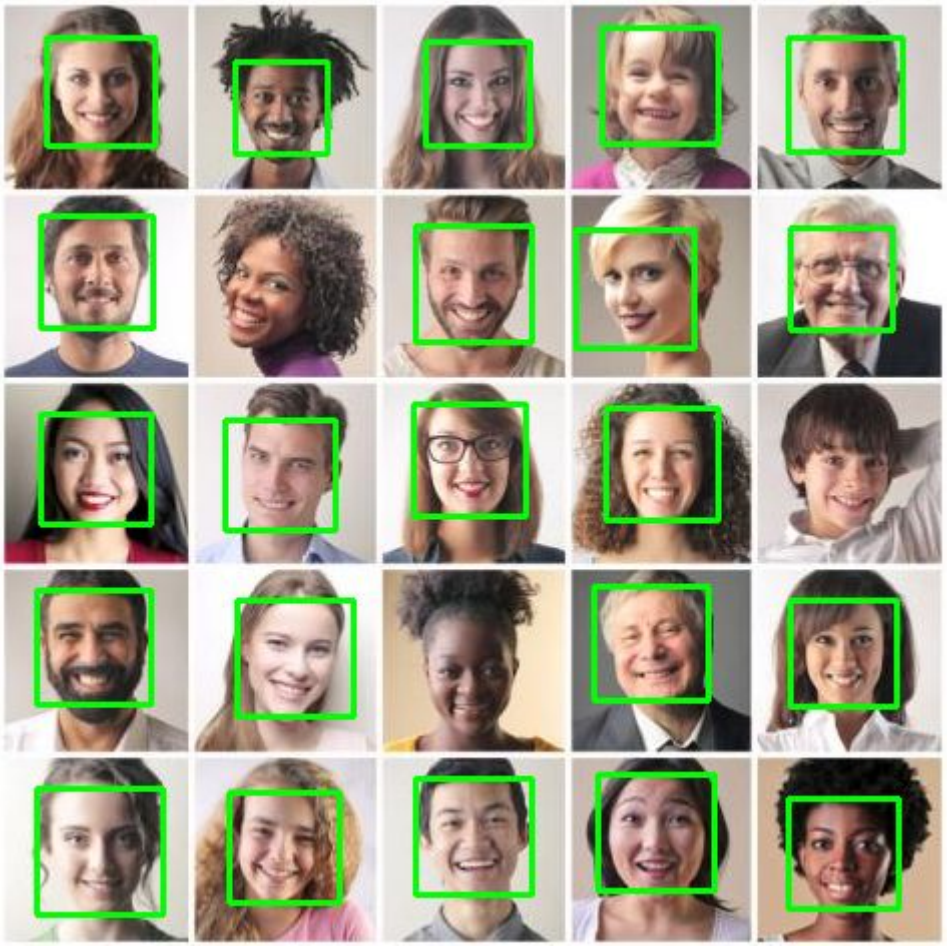


Question 1:

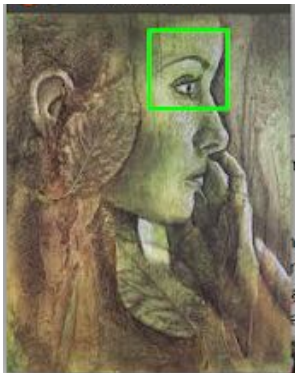
La cascade de Haar est une cascade de classificateurs permettant de détecter de manière rapide des objets (ici, des visages). Le terme cascade signifie que le classificateur final est le résultat de plusieurs classificateurs plus simples (que l'on appelle stages) qui sont appliqués successivement sur une région d'intérêt jusqu'à ce qu'un des stages échoue où qu'ils soient tous validés. Le premier classificateur est le plus optimisé et permet de rejeter plus rapidement une zone si l'objet recherché ne s'y trouve pas. Si potentiellement l'objet s'y trouve, alors le deuxième classificateur est utilisé et ainsi de suite jusqu'au dernier.
Chaque classificateur a en entrée des caractéristiques pseudo-Haar (Haar-like features en anglais) positionnée et dimensionné spécifiquement dans la zone d'intérêt. Ces caractéristiques doivent leur nom à leur similarité avec les ondelettes de Haar. Pour faire simple, on calcule la somme de pixels délimités par la zone sombre soustrait à la somme des pixels délimités par la zone claire. Cette valeur est celle qui est ensuite utilisée par le classificateur.

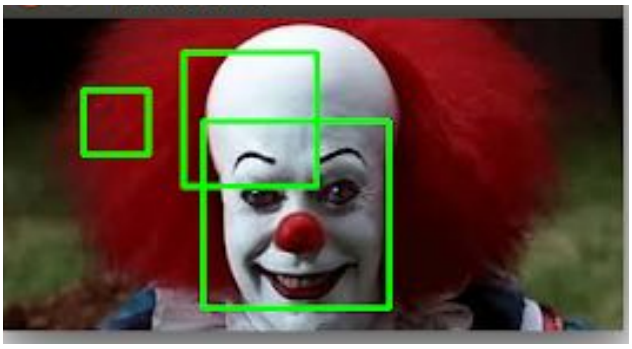
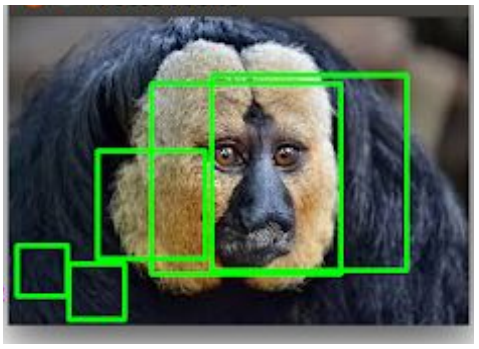
Détection de visage:
application des caractéristiques pseudo-Haar (viola 2001)

Dans l'exemple fourni, le classificateur fourni a été entraîné sur des visages d'un perspective frontale. En effet, nous remarquons sur ce test que le programme reconnaît tous les visage face à la caméra mais pas ceux legerement de profile ou inclinés

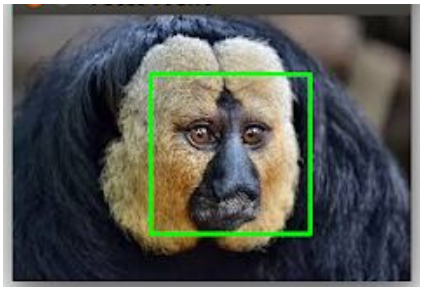


On va tester et éprouver les paramètres du code face_detect.py avec différentes sortes d'image : On va étudier l'influence des paramètres de l'appel à la cascade, notamment "scaleFactor", ici les images sont zoomées avec un petit scale (1.001) le script va identifier lplusiers visages qui n'en sont pas.



Tandis qu'avec un scale plus grand (1.2) le script marche “mieux” pour nos images et il identifie les vrais visages :

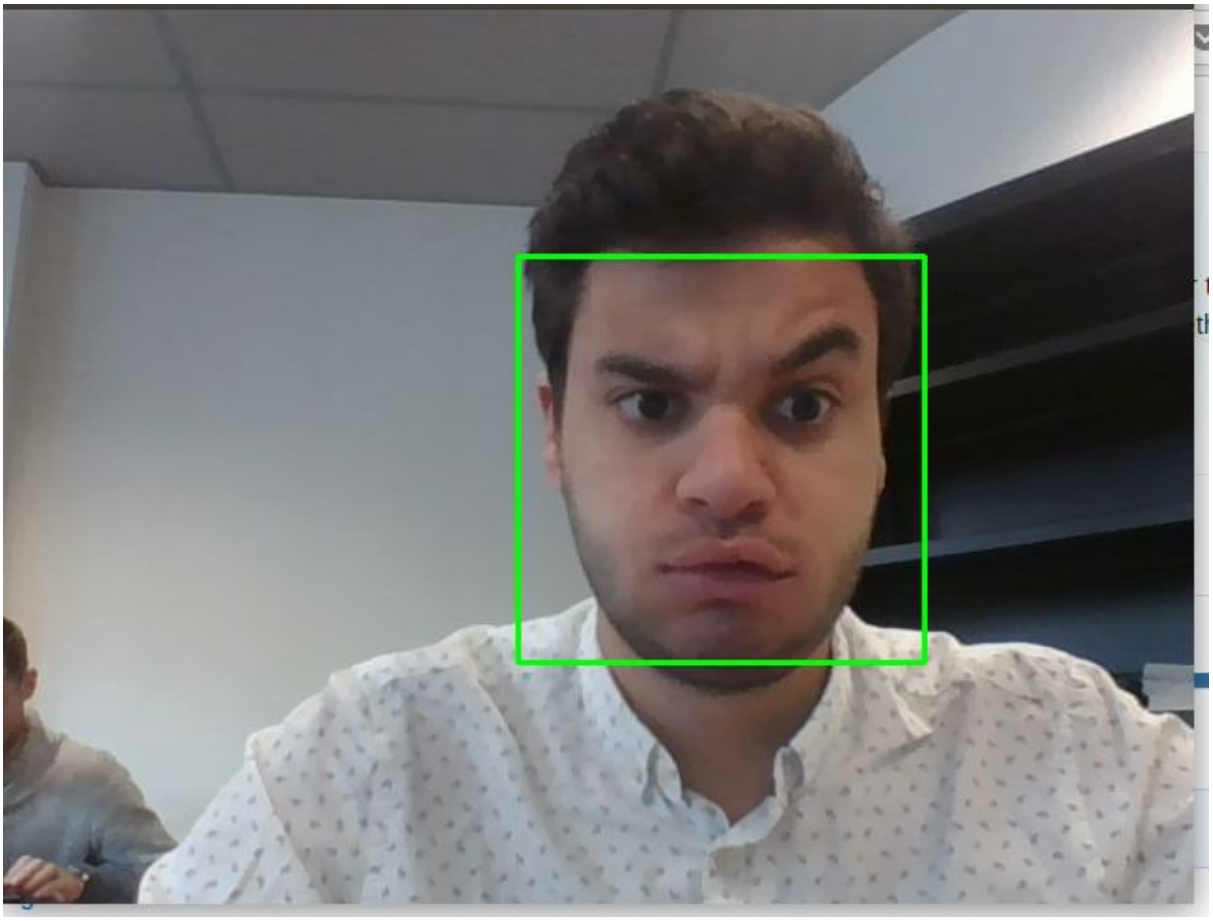




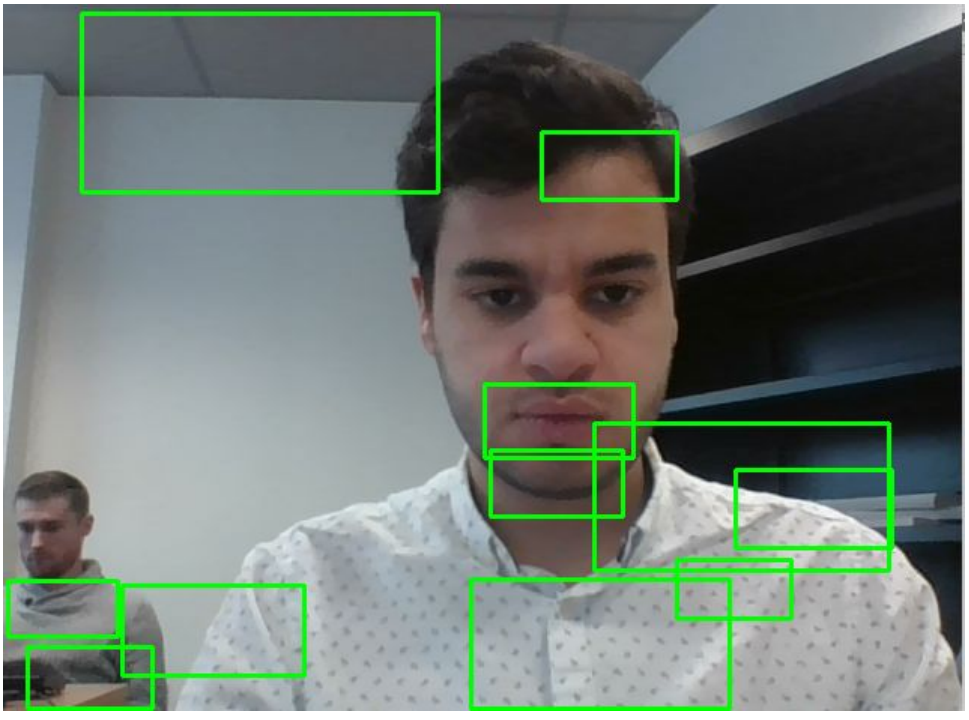
On voit donc qu'avec un “scaleFactor” plus adapté à la taille des visages sur l’image on arrive à obtenir une bonne identification des visages. Cependant pour des visages de profil et de moins bonne qualité le script ne fonctionne pas aussi bien.

Question 2:

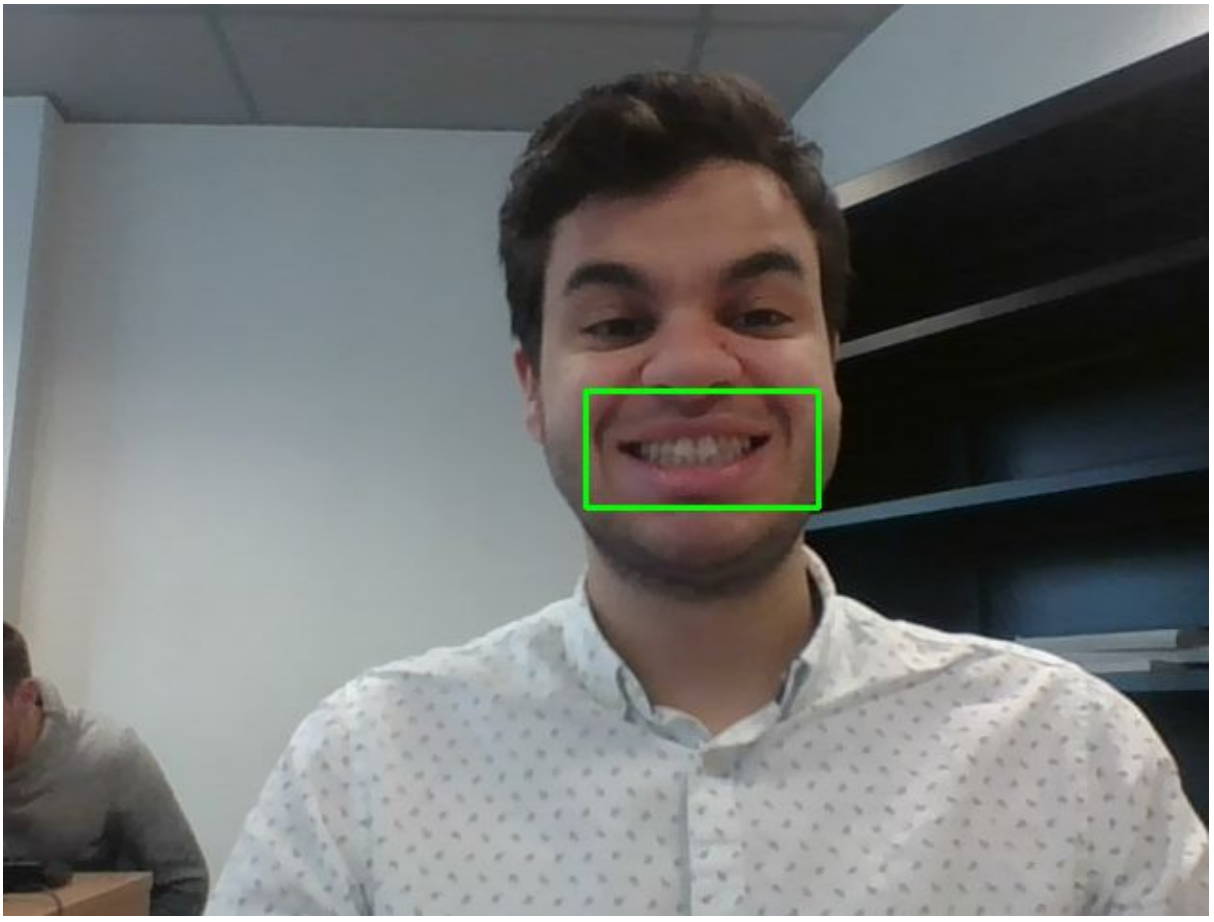
Dans cette partie il nous est demandé de faire des traitements en temps réel sur le flux vidéo de la webcam sous opencv. Pour cela nous utiliserons la fonction d’opencv “VideoCapture(X)” qui nous permet de sélectionner un périphérique vidéo (/dev/videoX, X étant le numéro du périphérique) et de venir lire son flux directement avec la fonction read(). Nous utilisons la même cascade de haar implémentée précédemment (haarcascade_frontalface_default.xml) affin de faire une detection de visage à travers la webcam.



Nous décidons par la suite d’essayer une autre cascade pour détecter les sourires, cependant avec les mêmes paramètre de détection que celle de front_face nous obtenons un resultat non satisfaisant:



Afin que notre programme marche nous remarquons qu'il faut adapter le minNeighbors et le scaleFactor à notre nouveau classificateur. Une fois nos modification faite, nous obtenons le résultat suivant:



Question 3:
Dans cette partie nous avons pour objectif d'apprendre comment générer nos propre classificateur, sur une base de données d'images de notre choix.
Pour former notre propre classificateur, nous avons besoin d'échantillons, ce qui signifie que nous avons besoin de beaucoup d'images qui montrent l'objet que nous voulons détecter (échantillon positif) et encore plus d'images sans cet objet (échantillon négatif).
Nous utilisons les images fournies dans le e-campus ou nous avons 28 images positives contenant des bananes et 898 images négative ne contenant aucune banane.
Afin de générer notre classificateur, il nous est dans un premier temps demandé de placer nos différentes images respectivement dans les dossier positives images et negative images., dont on listera les chemins d'acces dans les fichiers positives.txt et negatives.txt.

Avec nos images positives et négatives en place, nous sommes prêts à générer des échantillons à partir d'eux, que nous utiliserons pour la formation elle-même. Il nous faut des échantillons positifs et négatifs. Heureusement, nous avons déjà les échantillons négatifs en place. Pour citer la documentation OpenCV sur les échantillons négatifs :

"Les échantillons négatifs sont prélevés sur des images arbitraires. Ces images ne doivent pas contenir

objets détectés. Les échantillons négatifs sont énumérés dans un fichier spécial. Il s'agit d'un dans lequel chaque ligne contient un nom de fichier image (par rapport au fichier du fichier de description) de l'image échantillon négative."

Cela signifie que notre fichier negatives.txt servira de liste d'échantillons négatifs. Mais nous avons toujours besoin d'échantillons positifs et il existe de nombreuses façons différentes de les obtenir, qui conduisent toutes à des résultats différents quant à la précision de notre classificateur formé.

Nous utiliserons un outil qu'OpenCV nous donne : opencv_createsamples. Cet outil offre plusieurs options pour générer des échantillons à partir d'images d'entrée et nous donne un fichier *.vec que nous pouvons ensuite utiliser pour former notre classificateur.

opencv_createsamples génère un grand nombre d'échantillons positifs à partir de nos images positives, en appliquant des transformations et des distorsions. Puisqu'on ne peut transformer qu'une seule image jusqu'à ce que ce ne soit plus une version différente, nous avons besoin d'un peu d'aide pour obtenir un plus grand nombre d'échantillons à partir de notre nombre relativement petit d'images en entrée.

Nous allons utiliser pour ce fait le script perl de Naotoshi avec nos 28 images positives de dimension en sortie 24x24:

```
perl bin/createsamples.pl positives.txt negatives.txt samples 28\
        "opencv_createsamples -bgcolor 0 -bgthresh 0 -maxxangle 1.1\
        -maxyangle 1.1 maxzangle 0.5 -maxidev 40 -w 24 -h 24"
```

Nous obtenons par la suite 28 échantillons que nous fusionnons en 1 seul fichier .vec.

Une fois notre échantillon prêt, nous pouvons lancer l’entrainemment de notre classificateur. OpenCV offre deux applications différentes pour former un classificateur Haar : opencv_haartraining et opencv_traincascade. Nous allons utiliser opencv_traincascade car il permet au processus de formation d'être multithreadé, réduisant ainsi le temps nécessaire pour terminer.

Nous allons passer à Opencv_traincascade les paramettres suivants: Nos échantillons positifs (samples.vec), nos images négatives, le répertoire de sortie: classifier de notre référentiel, et la taille des échantillons (-w et -h). -numNeg spécifie combien d'échantillons négatifs sont présents et -precalcValBufSize et -precalcIdxBufSize combien de mémoire il faut utiliser pendant l'entraînement. -numPos devrait être inférieur à celui des échantillons positifs que nous avons générés.

```
|   3|        1| 0.569677|
+----+---------+---------+
|   4|        1|  0.35786|
+----+---------+---------+
END>
Training until now has taken 0 days 0 hours 9 minutes 8 seconds.

===== TRAINING 12-stage =====
<BEGIN
POS count : consumed   28 : 28
NEG count : acceptanceRatio    897 : 5.73455e-06
Precalculation time: 4
+----+---------+---------+
|  N |    HR   |    FA   |
+----+---------+---------+
|   1|        1|        1|
+----+---------+---------+
|   2|        1|        1|
+----+---------+---------+
|   3|        1| 0.686734|
+----+---------+---------+
|   4|        1| 0.439242|
+----+---------+---------+
END>
Training until now has taken 0 days 0 hours 17 minutes 1 seconds.

===== TRAINING 13-stage =====
<BEGIN
POS count : consumed   28 : 28
NEG count : acceptanceRatio    897 : 3.06563e-06
Precalculation time: 4
+----+---------+---------+
|  N |    HR   |    FA   |
+----+---------+---------+
|   1|        1|        1|
+----+---------+---------+
|   2|        1|        1|
+----+---------+---------+
|   3|        1| 0.561873|
+----+---------+---------+
|   4|        1| 0.561873|
+----+---------+---------+
|   5|        1| 0.344482|
+----+---------+---------+
END>
Training until now has taken 0 days 0 hours 32 minutes 56 seconds.

===== TRAINING 14-stage =====
<BEGIN
POS count : consumed   28 : 28
NEG count : acceptanceRatio    25 : 9.16995e-07
Required leaf false alarm rate achieved. Branch training terminated.
ibrahim@ibrahim:~/vision/TP2/opencv-haar-classifier-training$
```

Résultat entrainement

Question 4 :

Pour conclure ce second TP vision, nous allons intégrer tout ce que l'on a appris durant ce TP sous ROS. Pour cela nous nous intéresserons au package usb_cam qui renvois notre flux vidéo sur different Topics.

NB: image_view ne marche pas sur melodic de base. pour fixer le bug:

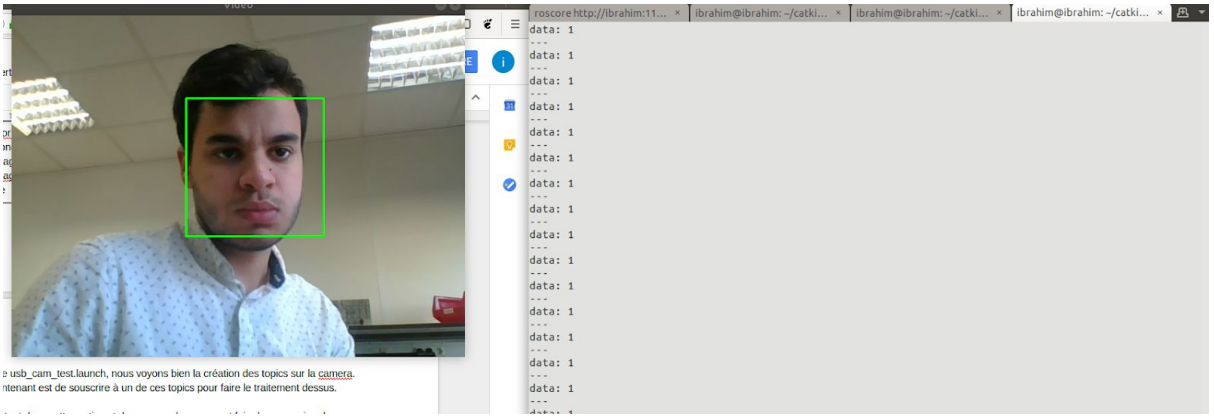
```
cd <your_workspace>
git -C src clone https://github.com/ros-perception/image_pipeline.git
git -C src/image_pipeline fetch origin pull/343/head:bugfix
git -C src/image_pipeline checkout bugfix
catkin_make
```

En lanssant le usb_cam_test.launch, nous voyons bien la création des topics sur la camera. L'objectif maintenant est de souscrire à un de ces topics pour faire le traitement dessus.

Le plus important dans cette partie est de comprendre comment faire la conversion du msg Image de sensors_msgs en frame d'opencv. Nous utiliserons pour cela CvBridge, qui est une bibliothèque ROS qui fournit une interface entre ROS et OpenCV. CvBridge se trouve dans le paquet cv_bridge dans la pile vision_opencv. Par la suite nous ferons exactement le même traitement fait dans la seconde partie de ce TP pour détecter des visages.

Enfin, il nous est demandé de créer un topic dans lequel nous allons publier un message Int16 correspondant au nombre de visage détectés. Nous récupérons ce nombre sur la longueur du tableau des visages détectés comme montré ci dessous:

```
faces = self.faceCascade.detectMultiScale(
gray,
scaleFactor=1.1,
minNeighbors=5,
minSize=(30, 30)
)
self.nb_face_pub.publish(len(faces))
```



rostopic echo /nb_face