Abdelouahed Nabil
N° 26893

DÉTECTION DU VISAGE 2D

Santé et prévention

Problématique et objectifs

- Coder différentes méthodes de détection de visage
  - Les tester sur une base de données d'images

➔ Quantifier les taux d'erreurs et les taux de détection

➔ Estimer les écarts entre les différentes techniques de détection

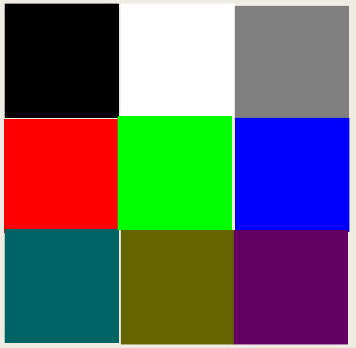
Introduction

- Une image = tableau de pixels
- 1 pixel = 3 octets ➔ RGB
- Exemple d'image :

[[[0,0,0],[255,255,255],[128,128,128]],

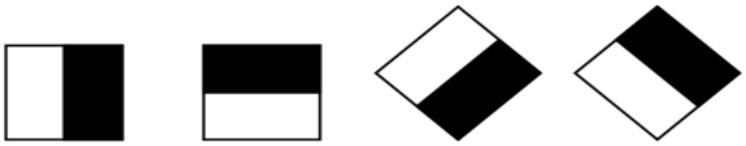
[[255,0,0],[0,255,0],[0,0,255]],

[[0,100,100],[100,100,0],[100,0,100]]]

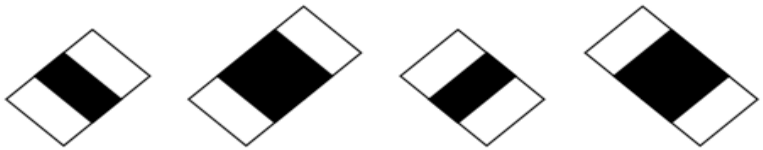


LA MÉTHODE VIOLA JONES (HAAR CASCADE)

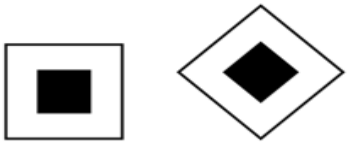
Caractéristiques de bord



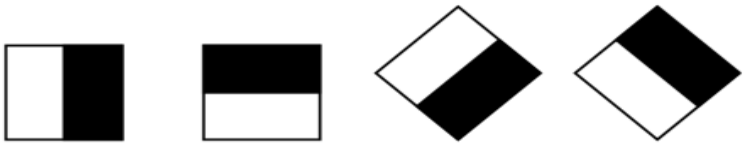
Caractéristiques de ligne



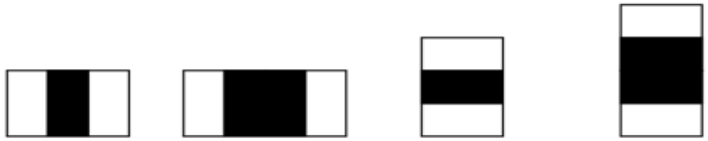
Caractéristiques centre-pourtour

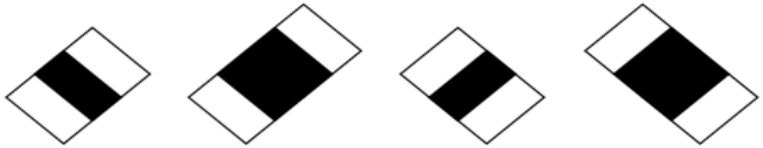


Caractéristiques de bord

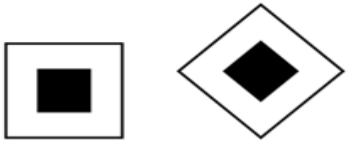


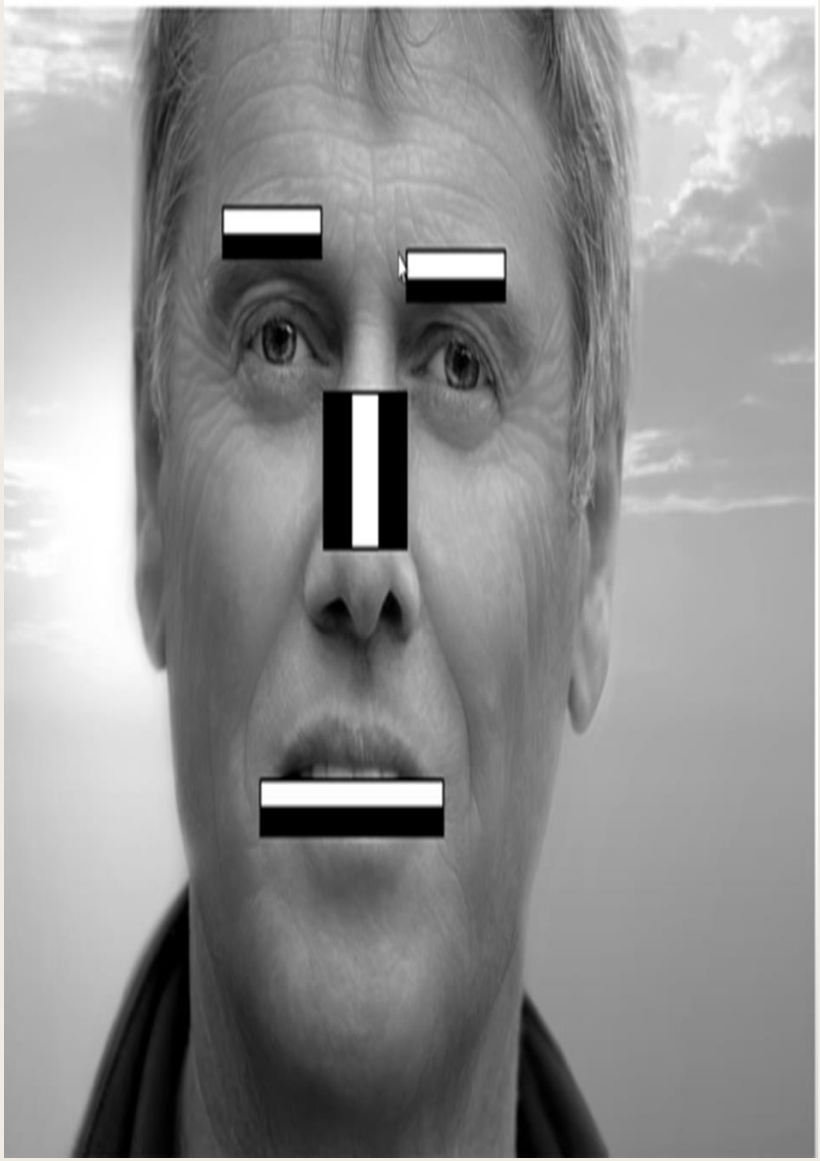
Caractéristiques de ligne



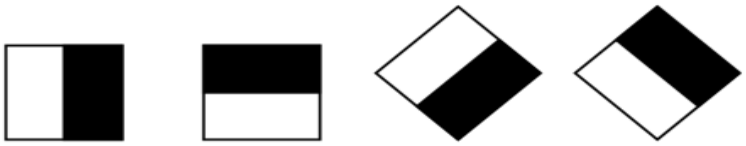


Caractéristiques centre-pourtour

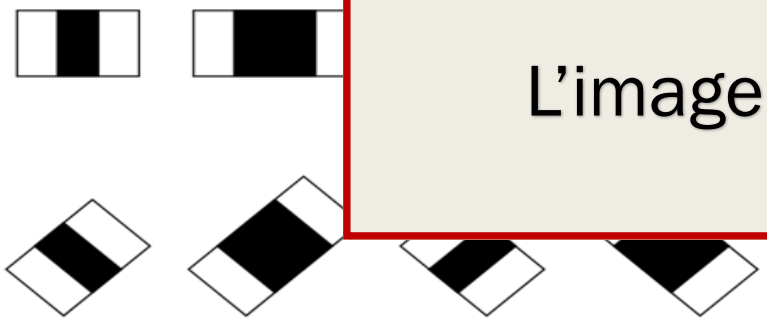




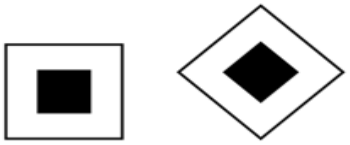
Caractéristiques de bord



Caractéristiques de ligne

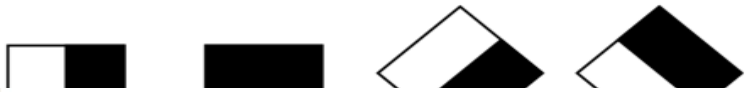


Caractéristiques centre-pourtour

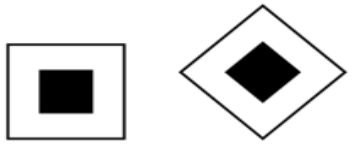


L'image intégrale

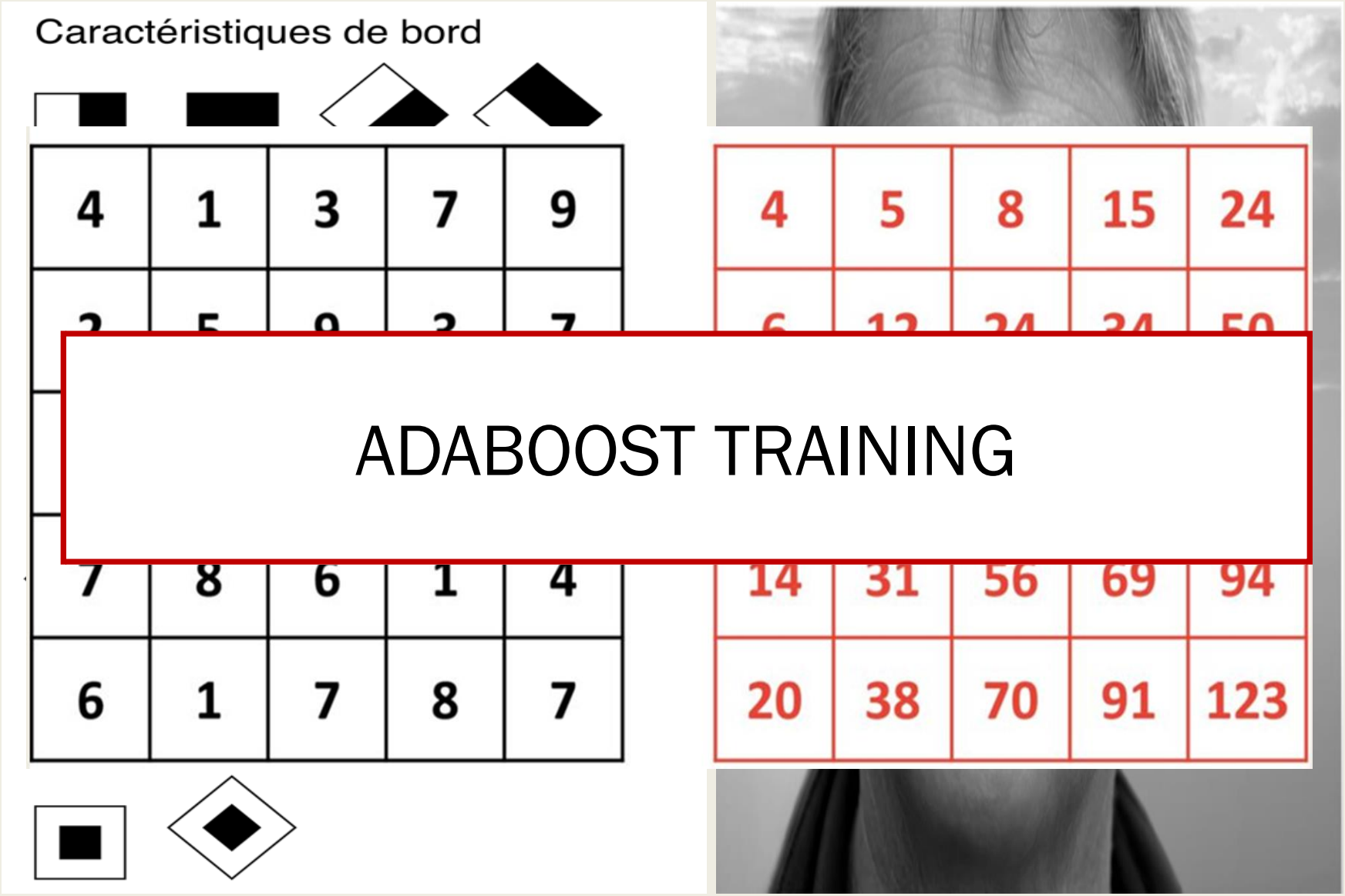
Caractéristiques de bord



4	1	3	7	9
2	5	9	3	7
1	3	7	2	5
7	8	6	1	4
6	1	7	8	7



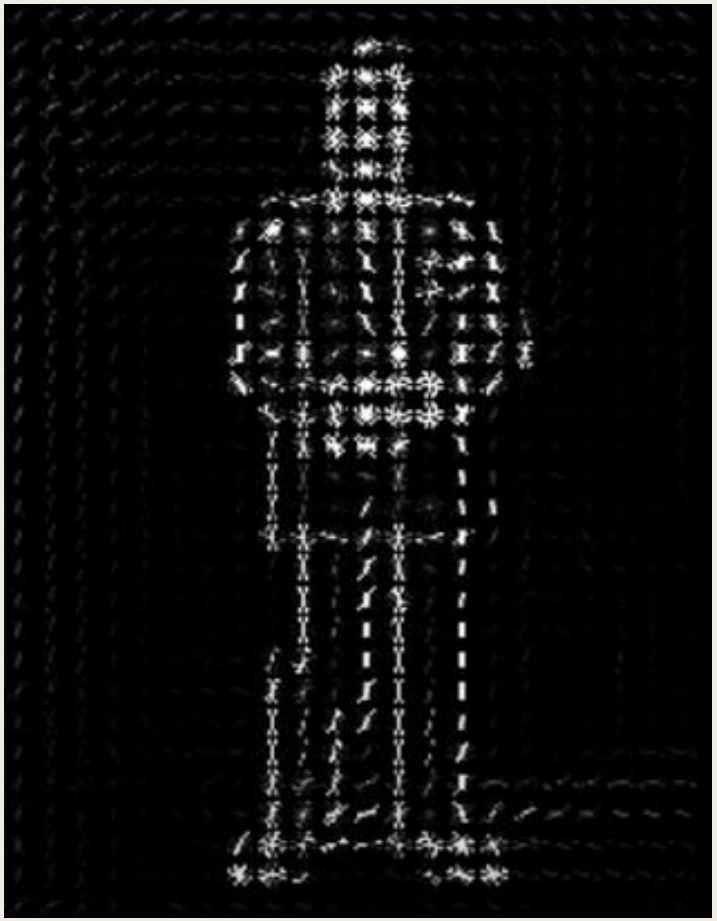
4	5	8	15	24
6	12	24	34	50
7	16	35	47	68
14	31	56	69	94
20	38	70	91	123


Caractéristiques de bord
4 1 3 7 9
7 8 6 1 4
6 1 7 8 7
4 5 8 15 24
14 31 56 69 94
20 38 70 91 123
ADABOOST TRAINING

HISTOGRAMME DE GRADIENT ORIENTÉ

Navneet Dalal et Bill Triggs

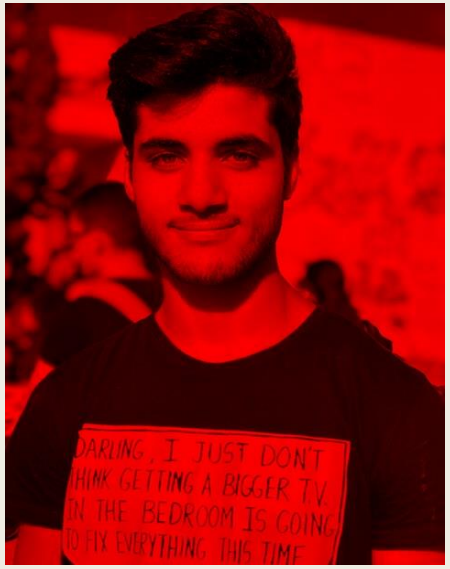
Exemple de HOG

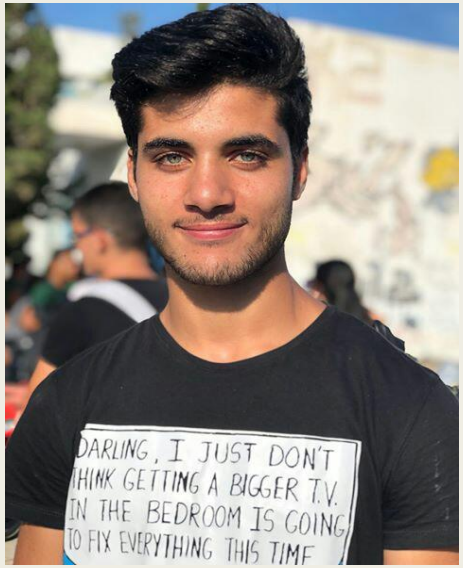


- Filtre dérivatif 1-D centré :

  $[\,-1\,,\,0\,,\,1\,]$ et $[\,-1\,,\,0\,,\,1\,]^T$

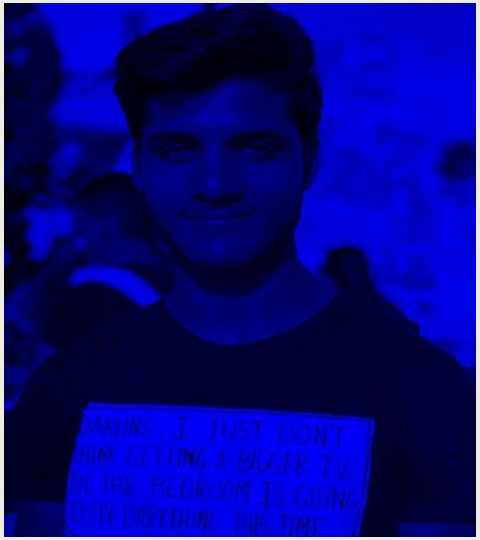
	100	
70	●	120
	50	

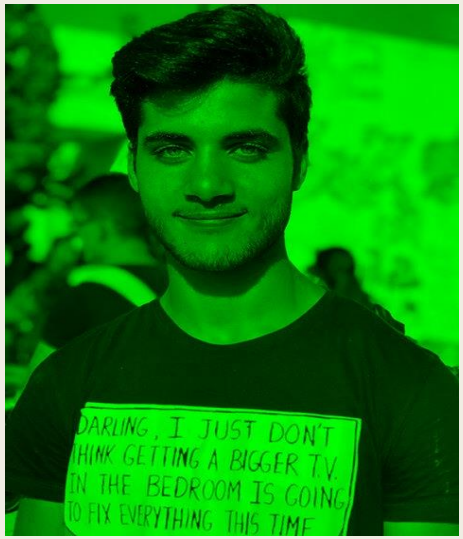




- <u>Norme du gradient</u> : $\sqrt{x^2 + y^2}$

- <u>Direction du gradient</u> : $\tan^{-1}\frac{y}{x}$



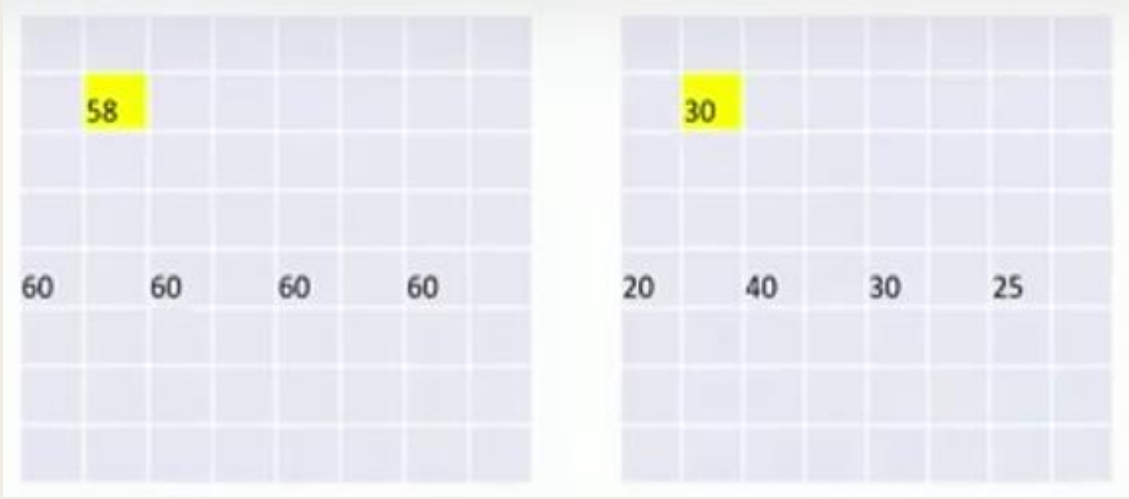


Réduire la taille des informations

Former les cellules 8 x 8 :

Un bloc : 64 vecteurs ➔ 9 valeurs

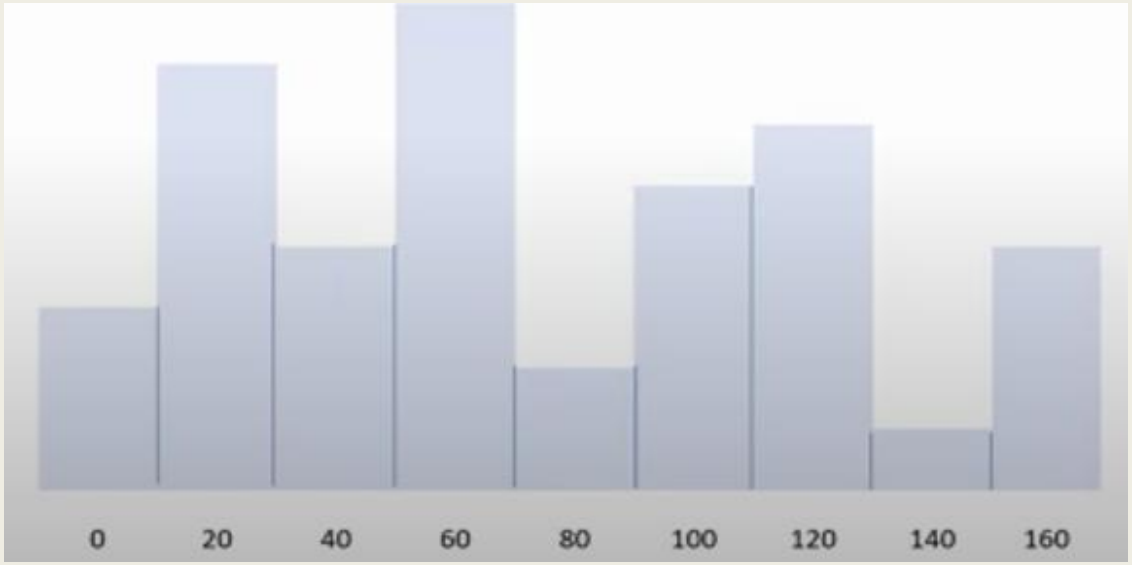
➔Normaliser les blocs !

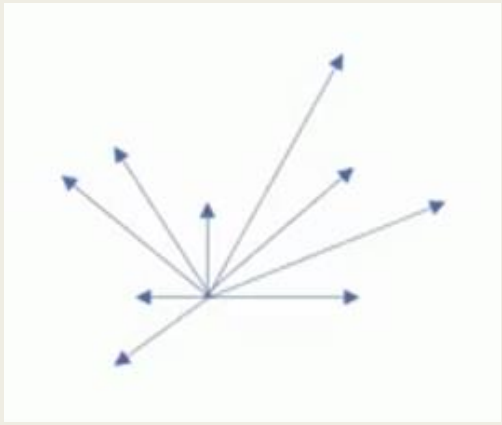


Norme du gradient

direction du gradient

$$\frac{180-0}{9} = 20$$





Vecteur caractéristique de taille 9

➔ VC =[50 , 135 , 105 , -----]

INTELLIGENCE ARTIFICIELLE ET BASE DE DONNÉES

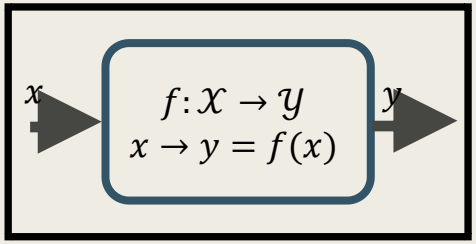
Les bases de données et l'IA

Les bases de données et l'IA

Phase 1

Apprentissage

Comportement observé (causes $\mathcal{X}_1$ et effets $\mathcal{Y}_1$)



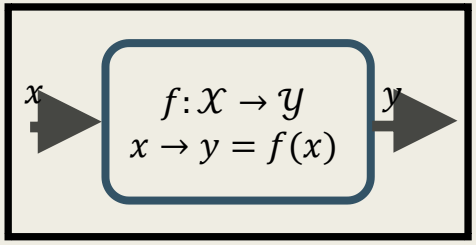
Modèle de comportement

Les bases de données et l'IA

Phase 1

Apprentissage

Comportement observé (causes $\mathcal{X}_1$ et effets $\mathcal{Y}_1$)

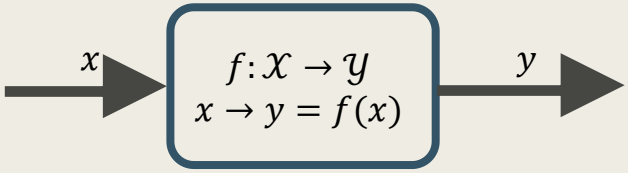


Modèle de comportement

Phase 2

Inférence

Causes $\mathcal{X}_2$



Modèle de comportement

$\mathcal{X}$

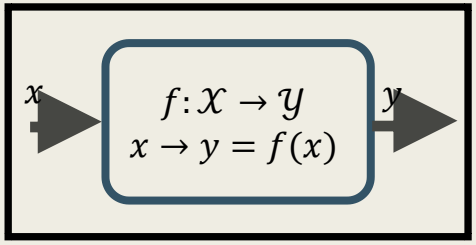
Effets $\mathcal{Y}_2$ prédis Par le modèle

Les bases de données et l'IA

Phase 1

Apprentissage

Base de données d'apprentissage $(\mathcal{X}_1, \mathcal{Y}_1)$

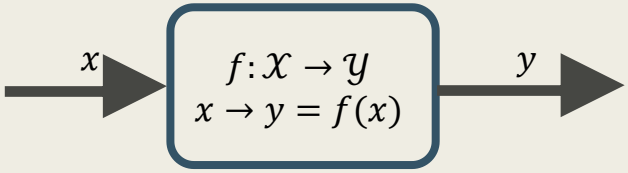


Modèle de comportement

Phase 2

Inférence

Causes $\mathcal{X}_2$



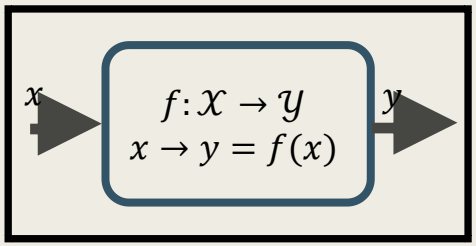
Modèle de comportement

$\mathcal{X}$

Effets $\mathcal{Y}_2$ prédis Par le modèle

Les bases de données et l'IA

Phase 1
Apprentissage

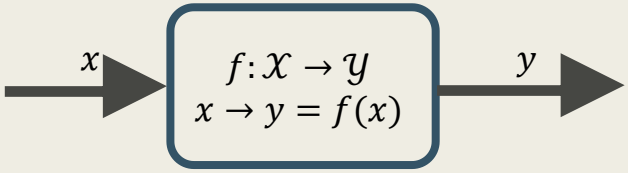


Base de données d'apprentissage $(\mathcal{X}_1, \mathcal{Y}_1)$

Modèle prédictif

Phase 2
Inférence

Causes $\mathcal{X}_2$



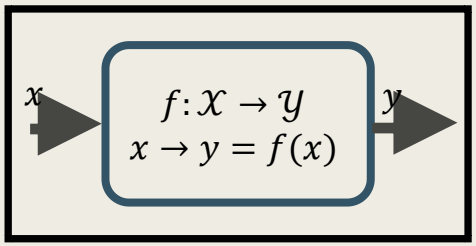
Modèle prédictif

$\mathcal{X}$

Effets $\mathcal{Y}_2$ prédis Par le modèle

Les bases de données et l'IA

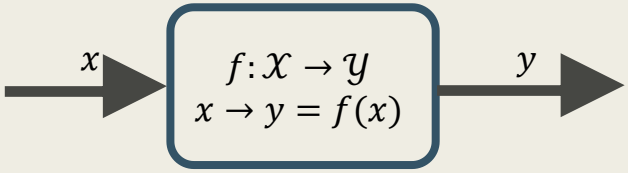
Phase 1
Apprentissage



Base de données d'apprentissage $(\mathcal{X}_1, \mathcal{Y}_1)$

Modèle prédictif

Phase 2
Inférence



Nouvelles données $\mathcal{X}_2$

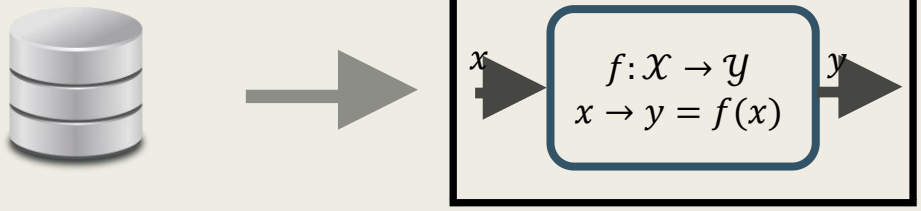
Modèle prédictif

$\mathcal{X}$

Effets $\mathcal{Y}_2$ prédis Par le modèle

Les bases de données et l'IA

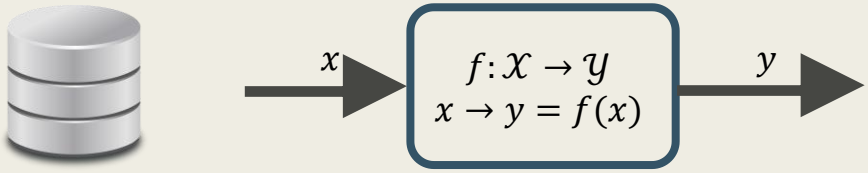
Phase 1
Apprentissage



Base de données d'apprentissage $(\mathcal{X}_1, \mathcal{Y}_1)$

Modèle prédictif

Phase 2
Inférence



Nouvelles données $\mathcal{X}_2$

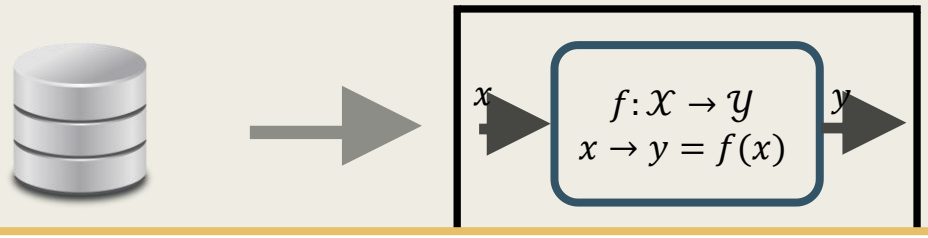
Modèle prédictif

$\mathcal{X}$

Sortie prédite $\mathcal{Y}_2$

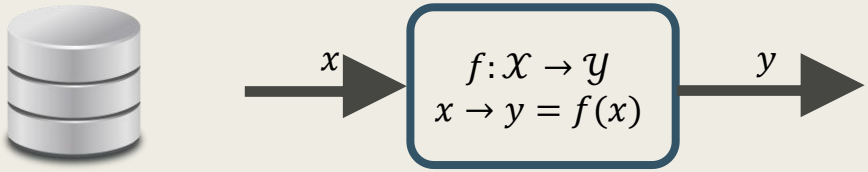
Les bases de données et l'IA

Phase 1
Apprentissage



l'apprentissage est fait en connaissant $\mathcal{X}$ et $\mathcal{Y}$: apprentissage supervisé

Phase 2
Inférence



Nouvelles données $\mathcal{X}_2$

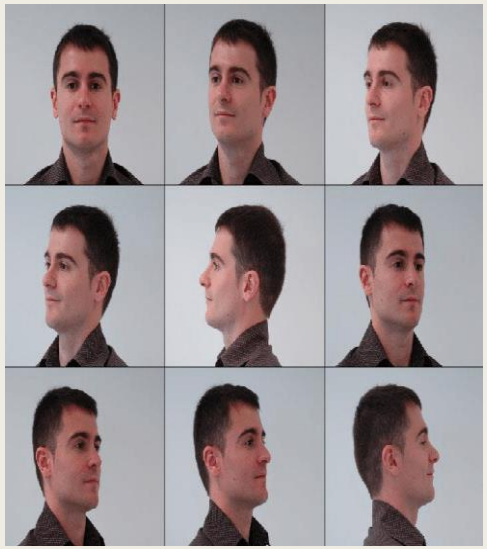
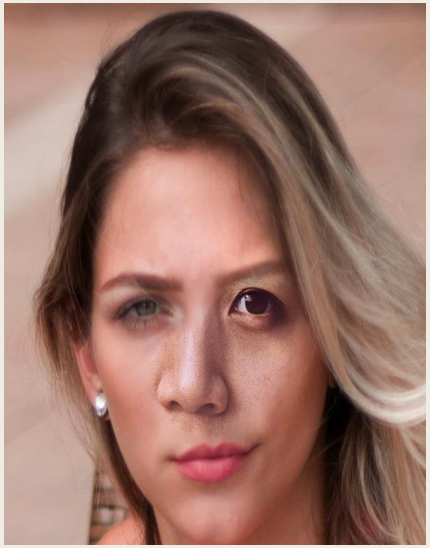
Modèle prédictif

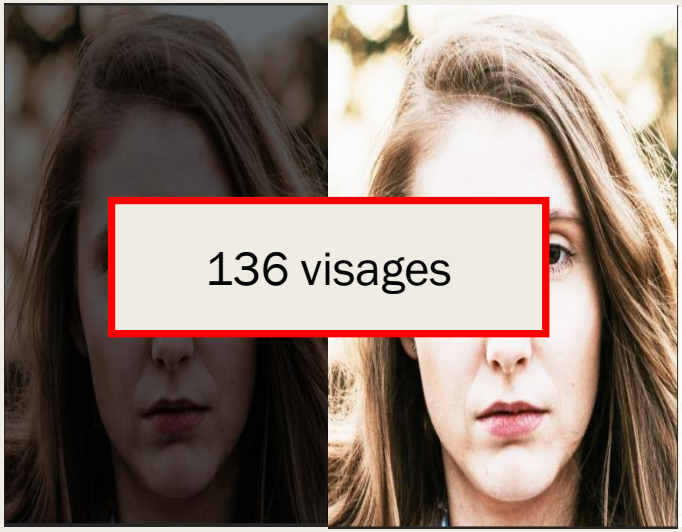
$\mathcal{X}$

Sortie prédite $\mathcal{Y}_2$







136 visages


34 visages


84 visages

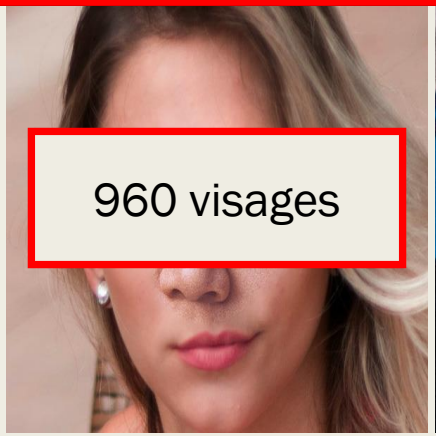

431 visages


20 visages


960 visages

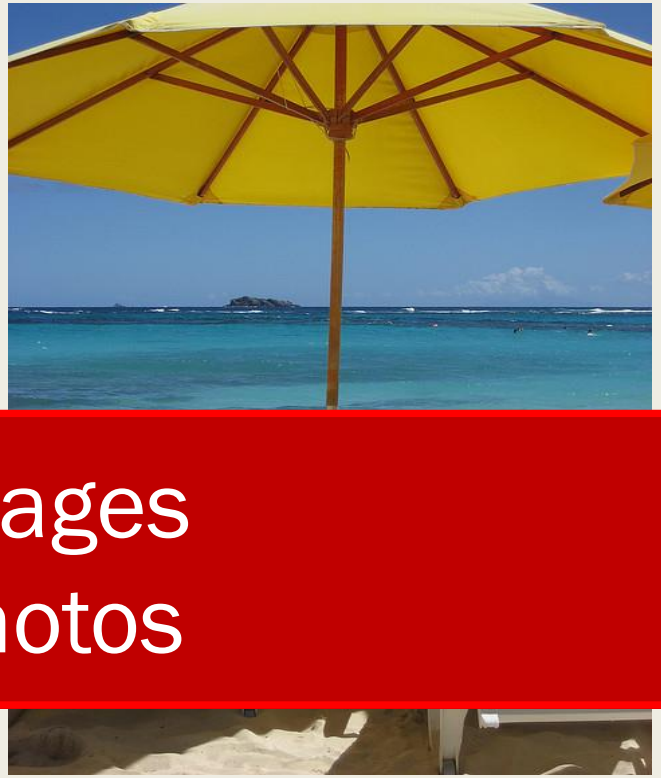

1081 visages
Base de donnée
de Google

2743 visages



1510 photos sans visages

Source : VQA : Visual Question Answering

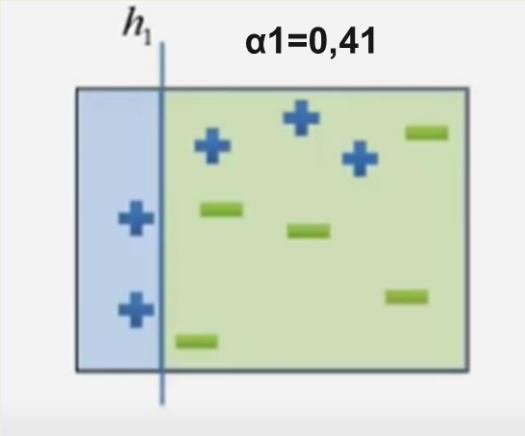


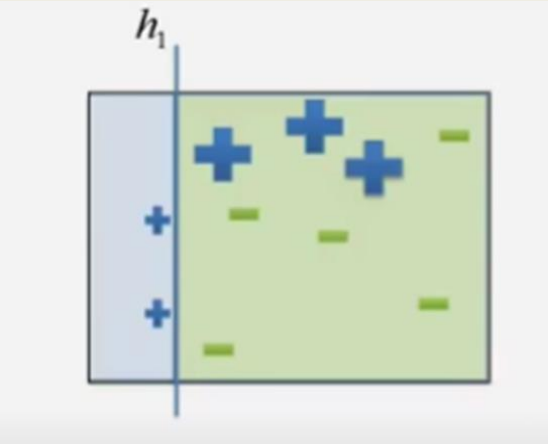
2743 visages
4253 photos

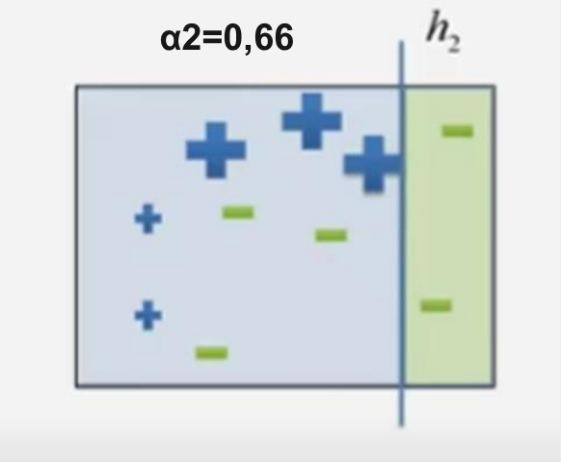
1510 photos sans visages

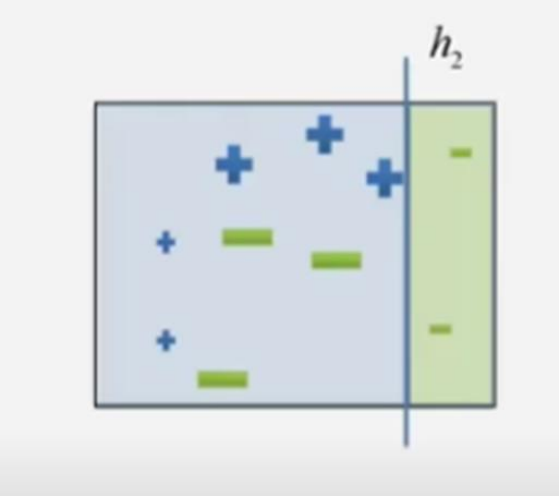
Source : VQA : Visual Question Answering

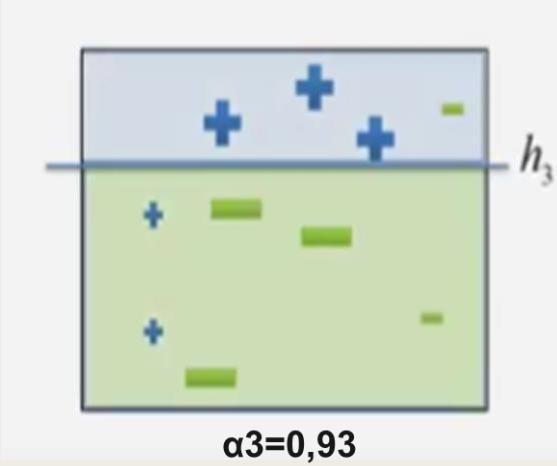
adaboost

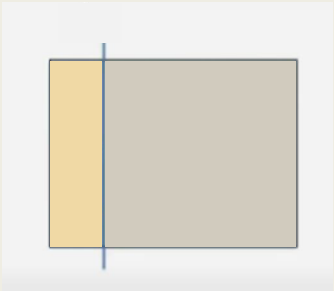
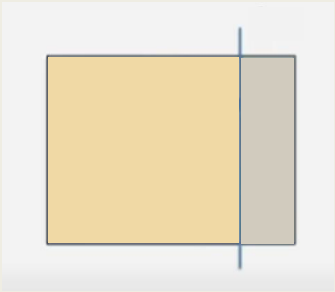
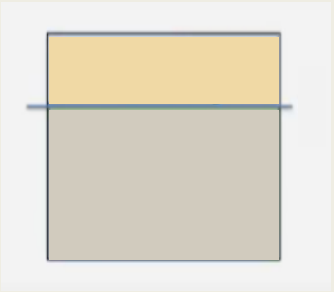


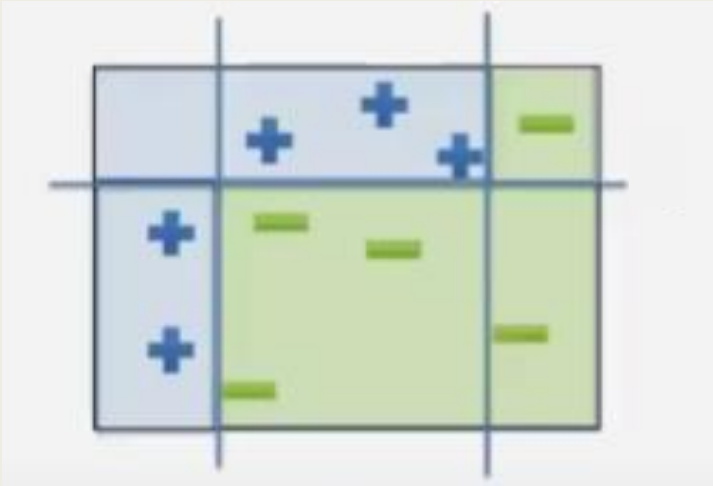






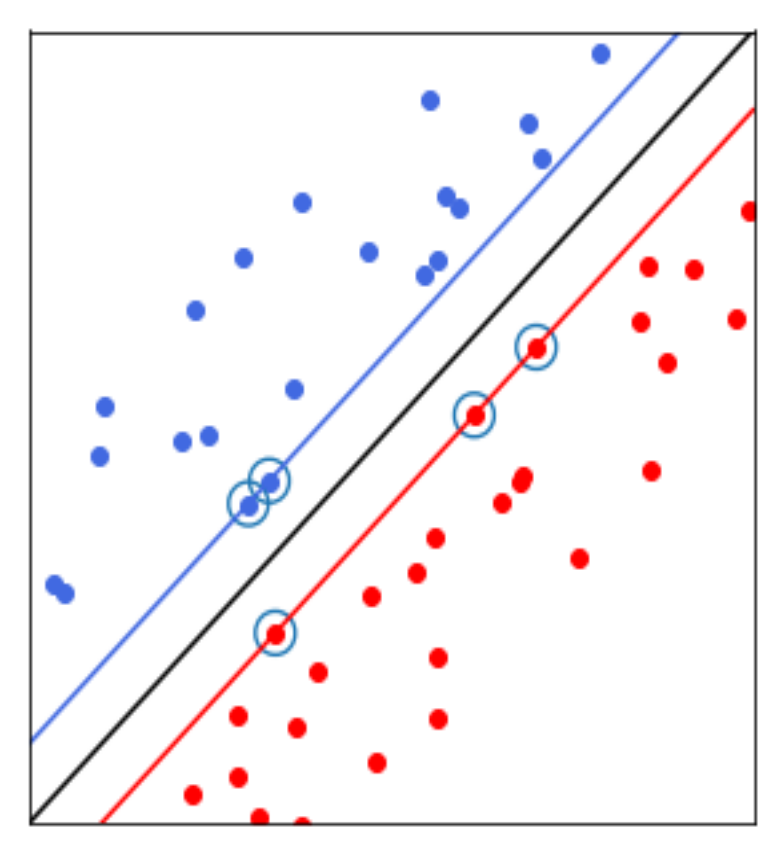


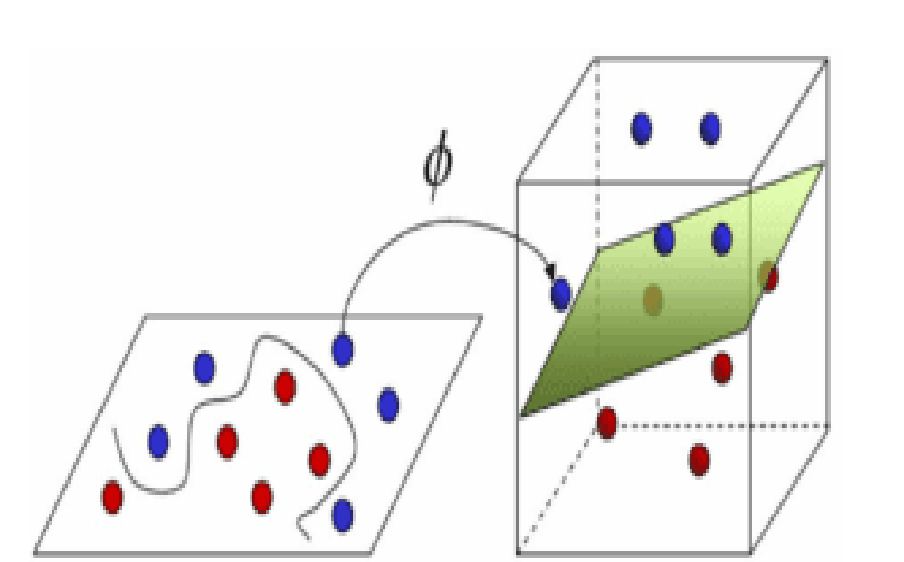
H = α1 *  + α2 *  + α3 * 

= 

Apprentissage :
images positives(4960 images)
et négatives (9544 images)

SVM





espace de redescription
(astuce du noyau)

le noyau Gaussien

$$e^{-\frac{||x-y||^2}{2\propto^2}}$$

le noyau polynomial

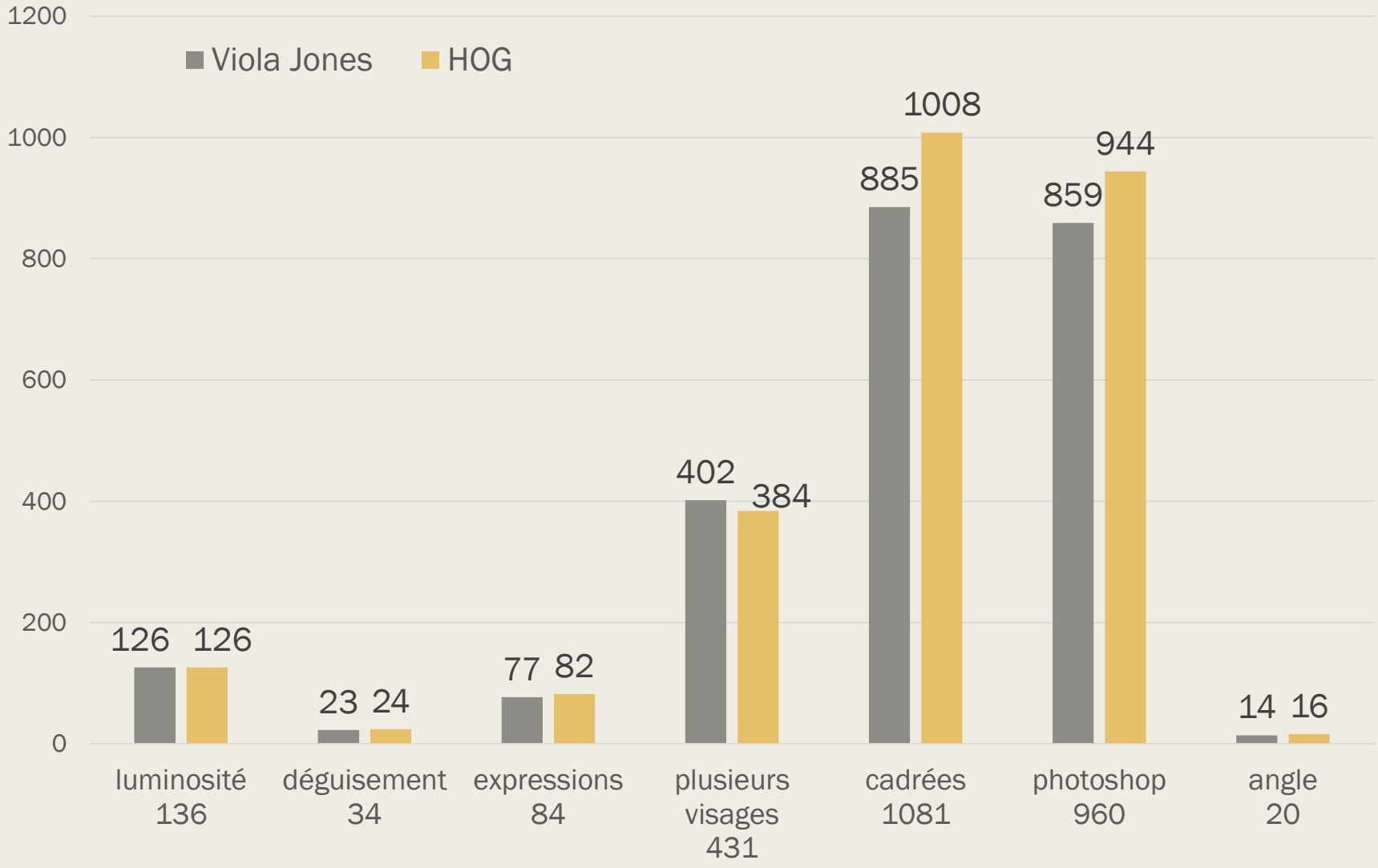
$$(< x, y > +1)^{d}$$

Source : DAP : Data Analytics Post

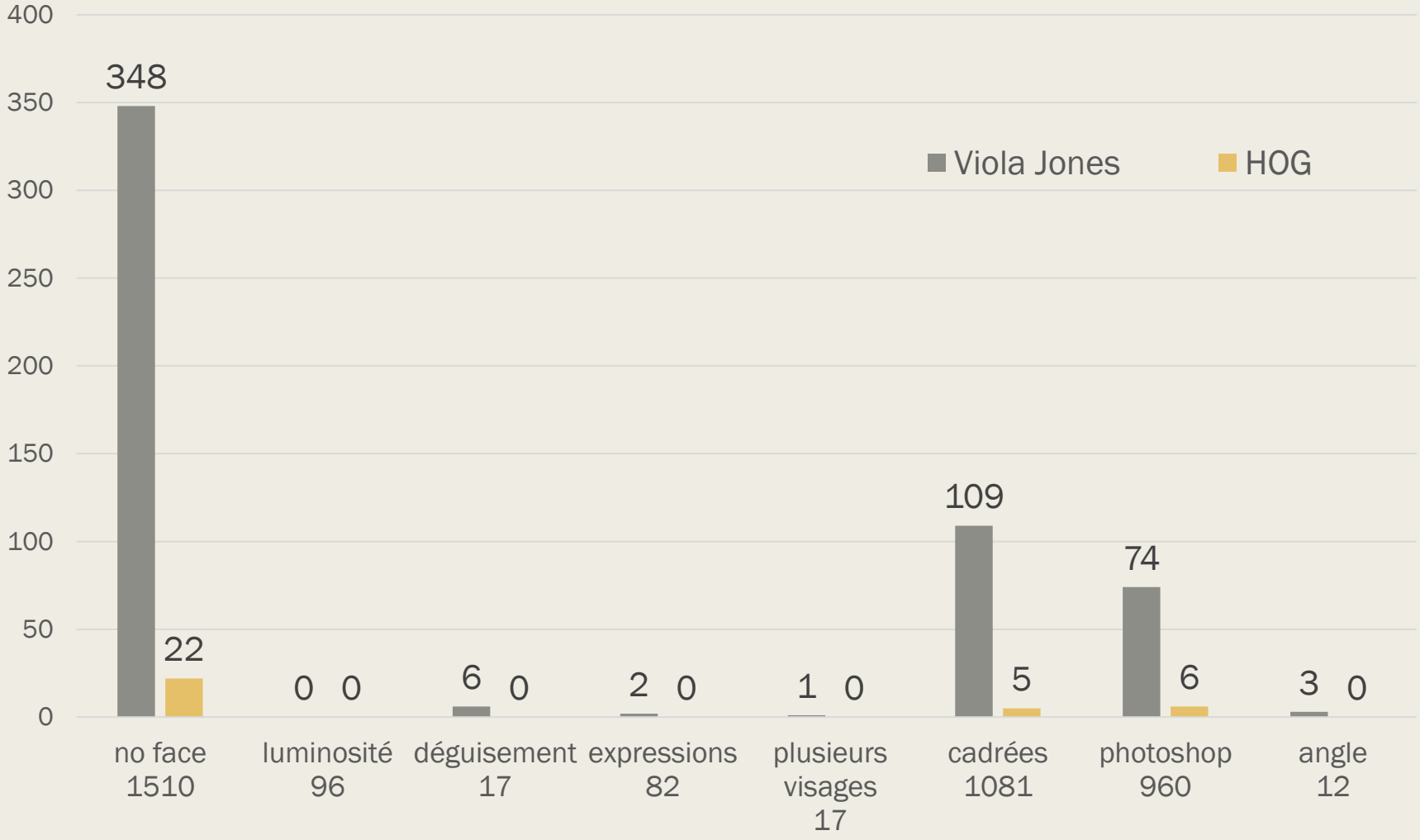
RÉSULTATS EXPÉRIMENTAUX

2743 visages
4253 photos

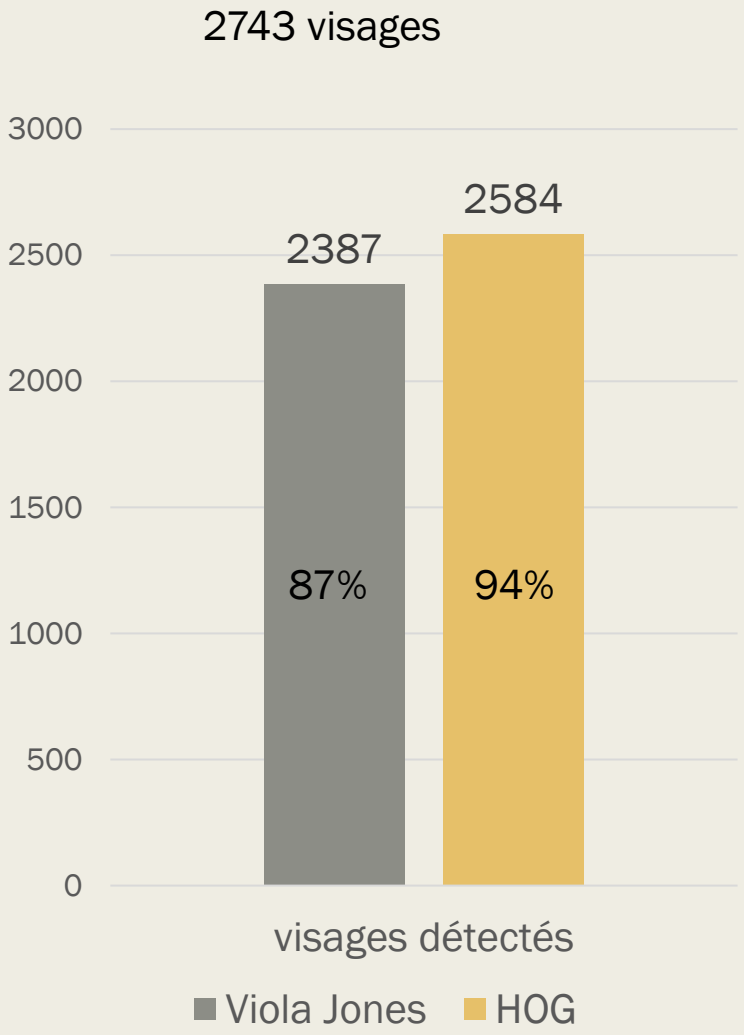
Visages détectés

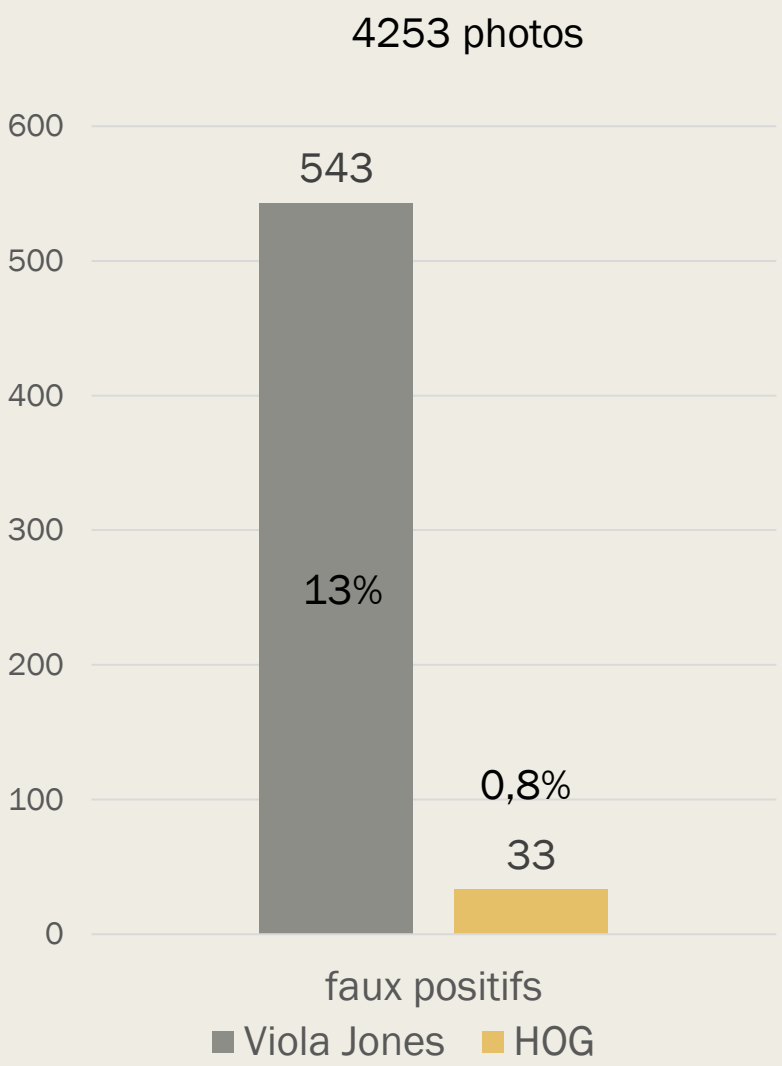


Les faux positifs



conclusion


2743 visages
3000
2500
2000
1500
1000
500
0
2387
2584
87%
94%
visages détectés
Viola Jones
HOG


4253 photos
600
500
400
300
200
100
0
543
13%
0,8%
33
faux positifs
Viola Jones
HOG

MERCI POUR VOTRE ATTENTION

```python
import cv2
#ouvrir l'image
img = cv2.imread('photos/obama twin.jpg')

# toutes l'image, toutes les lignes, les iémes pixels (0,1,2)
R, G, B = img[:,:,0], img[:,:,1], img[:,:,2]
#les coeffs : les poids de chaque couleur (sensitivité à la lumiére....)
imgGray = 0.2989 * R + 0.5870 * G + 0.1140 * B

#enregistrer l'image
cv2.imwrite('img.jpg',imgGray)
image_sortie=cv2.imread('img.jpg')
cv2.imshow('img.jpg',image_sortie)
cv2.waitKey(0)
```

```python
def image_int(img):
        #créer l'image intégrale avec des zéros
        img_integrale= [[0 for i in range(len(img[0]))] for h in range(len(img))]
        #calcul de l'image intégrale
        #k :les lignes de la matrice
        #j : les colonnes de la matrice
        for k in range(len(img)):
            for j in range(len(img[0])):
                if k==0 and j!=0:
                    img_integrale[k][j]+=img_integrale[k][j-1]+img[k][j]
                elif j==0 and k!=0 :
                    img_integrale[k][j]+=img_integrale[k-1][j]+img[k][j]
                else:
                    img_integrale[k][j]+=img[k][j]+img_integrale[k-1][j]+\
                                         img_integrale[k][j-1]-img_integrale[k-1][j-1]
        return(img_integrale)
```

```python
import cv2
# load a cascade for face detection
#xml : store and transport data
face_cascade=cv2.CascadeClassifier('folder/cascades/haarcascade_frontalface_default.xml')
eye_cascade=cv2.CascadeClassifier('folder/cascades/haarcascade_eye.xml')
#charger l'image
img=cv2.imread('photos/obama twin.jpg')
#convertir l'image en grayscale car cascade classifier expects grayscale imgs
gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
#avec cette ligne on va detecter le visage
faces=face_cascade.detectMultiScale(gray,1.08,5)
for (x,y,w,h) in faces:
    #dessiner le rectangle autour du visage
    #x représente left right coordonnées (horizontale)
    #y représente up down coordonnées (verticale)
    #w largeur du rectangle
    #h hauteur du rectange
    img=cv2.rectangle(img,(x,y),(x+w, y+h),(255,0,0),2)
cv2.namedWindow('obama detected')
cv2.imshow('obama detected',img)
cv2.imwrite('obama detected.jpg',img)
cv2.waitKey(0)
```

```python
import cv2


# load a cascade for face and eyes detection
face_cascade=cv2.CascadeClassifier('folder/cascades/haarcascade_frontalface_default.xml')
eye_cascade=cv2.CascadeClassifier('folder/cascades/haarcascade_eye.xml')
#activer la cam jusqu'a ce que l'utilisateur appui sur une touche
camera=cv2.VideoCapture(0)
while (cv2.waitKey(1)==-1):
    success, frame =camera.read()
    if success :
        gray=cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
        #jusqu'a ici chaque image (frame) capturée avec succes est converti en grayscale
        #detection du visage
        faces=face_cascade.detectMultiScale(gray,1.3,5)
        for (x,y,w,h) in faces:
            cv2.rectangle(frame,(x,y),(x+w, y+h),(255,0,0),2)
            #dans le rectangle dessiné autour du visage on detecte les yeux et on dessine un rectange autour chaque oeil
            roi_gray=gray[y:y+h, x:x+w]
            eyes=eye_cascade.detectMultiScale(roi_gray,1.03,5)
            for (ex,ey,ew,eh) in eyes:
                cv2.rectangle(frame,(x+ex,y+ey),(x+ex+ew,y+ey+eh),(0,255,0),2)
    cv2.imshow('face detection',frame)
```

```python
def gradient(img):
    # séparer les 3 composantes de chaque pixel
    R, G, B = img[:, :, 0], img[:, :, 1], img[:, :, 2]
    # créer les matrices vides pour les gradients horizontaux et verticaux pour chaque composante du pixel
    xr, xg, xb, yr, yg, yb = np.zeros(np.shape(R)), np.zeros(np.shape(R)), np.zeros(np.shape(R)), np.zeros(
        np.shape(R)), np.zeros(np.shape(R)), np.zeros(np.shape(R))
    # initialiser la première matrice des gradients verticaux (1ére et dernière ligne)
    for p in range(len(R)):
        yr[0], yg[0], yb[0] = -R[1], -G[1], -B[1]
        yr[-1], yg[-1], yb[-1] = R[-2], G[-2], B[-2]
    # initialiser la première matrice des gradients horizontaux (1ére et dernière colonne)
    for k in range(len(R)):
        xr[k][0], xg[k][0], xb[k][0] = R[k][1], G[k][1], B[k][1]
        xr[k][-1], xg[k][-1], xb[k][-1] = -R[k][-2], -G[k][-2], -B[k][-2]
    for i in range(len(R)): # calcul des gradients horizontaux
        for j in range(1,len(R[1])-1):
            xr[i][j] = R[i][j + 1] - R[i][j - 1]
            xg[i][j] = G[i][j + 1] - G[i][j - 1]
            xb[i][j] = B[i][j + 1] - B[i][j - 1]
    # calcul des gradients verticaux
    for h in range(1,len(R)-1):
        for n in range(len(R[1])):
            yr[h][n] = R[h-1][n] - R[h+1][n]
            yg[h][n] = G[h-1][n] - G[h+1][n]
            yb[h][n] = B[h - 1][n] - B[h + 1][n]
    return(xr,xg,xb,yr,yg,yb)
```

```python
# calcul de la norme de chaque gradient
def magnitude_gradient(xr,xg,xb,yr,yg,yb):
    mag = np.zeros(np.shape(xr))
    grad_x, grad_y = np.zeros(np.shape(xr)), np.zeros(np.shape(yr))
    mag_r = np.sqrt(xr ** 2 + yr ** 2)
    mag_g = np.sqrt(xg ** 2 + yg ** 2)
    mag_b = np.sqrt(xb ** 2 + yb ** 2)
    # garder le gradient de plus grande norme entre les trois composnates R G B
    for i in range(len(mag)):
        for j in range(len(mag[1])):
            mag[i][j]=max(mag_r[i][j],mag_g[i][j],mag_b[i][j])
            if mag[i][j]==mag_r[i][j]:
                grad_x[i][j], grad_y[i][j] = xr[i][j], yr[i][j]
            elif mag[i][j] == mag_g[i][j]:
                grad_x[i][j], grad_y[i][j] = xg[i][j], yg[i][j]
            else:
                grad_x[i][j], grad_y[i][j] = xb[i][j], yb[i][j]
    return(mag,grad_x,grad_y)
```

```python
# calcul de l'angle du gradient en degrés
def angle_gradient(grad_x,grad_y):
    # le signe de l'angle n'est pas necessaire pour la détection du visage
    angle = abs(np.arctan(grad_y/grad_x))
    angle = angle*(180/np.pi)
    return (angle)
```

```python
# calcul de la matrice pour les histogrammes des blocs 8*8
def mat_histogramme(mag,angle):
    # créer un tableau de 8*16 listes à 9 valeurs
    histo = np.array([[[0]*9 for i in range(len(mag[1])//8)] for j in range(len(mag)//8)])
    # remplir la matrice
    L=[0,20,40,60,80,100,120,140,160,180]
    for h1 in range(len(angle)):
        for h2 in range(len(angle[h1])):
            if angle[h1][h2]==180:
                angle[h1][h2] = 0
    for i in range(0,len(mag),8):
        for j in range(0,len(mag[i]),8):
            for k in range(8):
                for p in range(8):
                    a = angle[i+k][j+p] // 20
                    b = a + 1
                    prop_min = (L[b] - angle[i+k][j+p]) / (L[b] - L[a])
                    prop_max = 1 - prop_min
                    histo[i // 8][j // 8][a] += mag[i + k][j + p] * prop_min
                    if b == 9:
                        b = 0
                    histo[i // 8][j // 8][b] += mag[i + k][j + p] * prop_max
            # normaliser les blocs
            s = 0
            for h1 in range(len(histo[i // 8][j // 8])):
                s += histo[i // 8][j // 8][h1]**2
            s =np.sqrt(s)
            for h2 in range(len(histo[i // 8][j // 8])):
                histo[i // 8][j // 8][h2] = histo[i // 8][j // 8][h2]/s
    return(histo)
```

```python
# représenter un histogramme
def histogramme(Lx,Ly):
    L=[]
    for i in range(len(Lx)):
        for j in range(round(Ly[i])):
            L.append(Lx[i])
    plt.hist(L, range=(0, 160), bins=Lx, color='blue',
             edgecolor='red')
    return(None)

# représenter les histogrammes pour tout les blocs 8*8
def histogramme_blocs(histo):
    L = [0, 20, 40, 60, 80, 100, 120, 140, 160]
    for i in range(len(histo)):
        for j in range(len(histo[i])):
            histogramme(L,histo[i][j])
    return(None)
```

```python
# représenter les histogrammes en vecteurs
def vecteurs(histo):
    L = [0,20,40,60,80,100,120,140,160]
    for i in range(len(histo)):
        for j in range(len(histo[i])):
            plt.plot(-1.5, -1.5, "None")
            plt.plot(1.5, 1.5, "None")
            plt.axis('equal')
            plt.grid()
            for k in range(9):
                x = np.cos(L[k]*np.pi/180) * histo[i][j][k]
                y = np.sin(L[k]*np.pi/180) * histo[i][j][k]
                plt.quiver(0, 0,x, y, angles = 'xy', scale_units = 'xy', scale = 1)
            plt.show()
    return(None)
```

```python
# programme final
def HOG(img):
    xr, xg, xb, yr, yg, yb = gradient(img)
    mag, grad_x, grad_y = magnitude_gradient(xr, xg, xb, yr, yg, yb)
    angle = angle_gradient(grad_x, grad_y)
    histo = mat_histogramme(mag, angle)
    histogramme_blocs(histo)
    vecteurs(histo)
    return(None)
```

```python
import face_recognition as fr
from PIL import Image, ImageDraw
import cv2, os, numpy as np

path1 = "D:/projects python pycharm/tipe/HOG/photos/test images/real_and_fake_face/training_real/"
path2 = "D:/projects python pycharm/tipe/HOG/photos/résultats/training_real/"
listing = os.listdir(path1)
for file in listing:
    #import l'image
    image = fr.load_image_file(path1+file)
    #localiser le visage
    loc = fr.face_locations(image)
    #dessiner un rectangle autour du visage
    #passer de tableau de pixel a image pour dessiner dessus
    image_pil = Image.fromarray(image)
    draw = ImageDraw.Draw(image_pil)

    for (x,y,z,t) in loc:
        draw.rectangle(((y,x),(t,z)), outline=(0,0,255))

    image_pil.save(path2+file)
```