UNIVERSIDAD SANTO TOMÁS

PRIMER CLAUSTRO UNIVERSITARIO DE COLOMBIA

SECCIONAL TUNJA

VIGILADA MINEDUCACIÓN - SNIES 1732

Acreditación Institucional Internacional

OTORGADA POR EL IAC CINDA ACUERDO 55 DEL 8 DE MAYO–VIGENCIA 5 AÑOS

ACREDITACIÓN INSTITUCIONAL DE ALTA CALIDAD MULTICAMPUS

Vigencia por seis años

QS STARS RATED FOR EXCELLENCE

ISO 9001 icontec SC4289-1

CERTIFIED IQNet MANAGEMENT SYSTEM

UNIVERSIDAD SANTO TOMÁS
PRIMER CLAUSTRO UNIVERSITARIO DE COLOMBIA
SECCIONAL TUNJA
VIGILADA MINEDUCACIÓN - SNIES 1732

Faculty: Systems engineer
Course: Deep Learning
Topic: Computer visión
SubTopic: Detección de rostros o caras frontales usando Haar (Clasificadores/cascades)

Professor: Luis Fernando Castellanos Guarin
Email: Luis.castellanosg@usantoto.edu.co
Phone: 3214582098

CONTENIDO

1. Reconocimiento de objetos.
2. Clasificador de cascada - Haar.
3. Usando clasificador Haar en Google colaboratory
4. Entrenar modelo para que reconozca rostros particulares

¡Siempre hacia lo alto!

“Detección"

VS

“Reconocimiento"

La detección de objetos:

Realiza una búsqueda especifica de un objeto en una imagen, pero sin entregar una información precisa.

El reconocimiento de objetos:

Además de haber pasado por una **detección** especifica de la imagen, también es posible entregar **información mas precisa**, como por ejemplo; si el rostro es de una mujer y si es así...de que mujer se trata (un familiar, famoso, conocido, etc.)

En la actualidad (**2020**) existen distintos algoritmos que pueden ser implementados para una detección facial.

Vamos a trabajar los métodos mas populares y eficientes que permiten detectar objetos en tiempo real.

Clasificador de Cascada

Mejor conocido como algoritmo o clasificador Haar, fue el primer framework de detección de objetos propuesto por Paul Viola y Michael Jones en 2001.

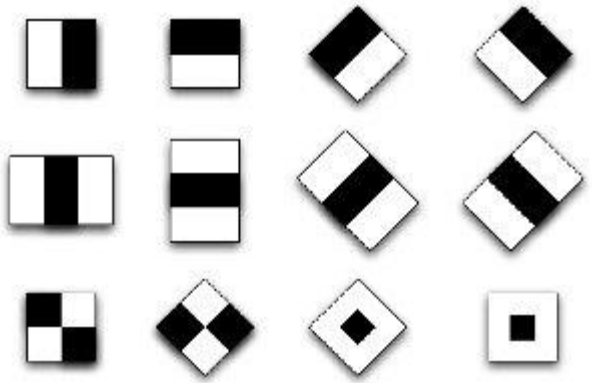
Se trata de un enfoque basado en el **aprendizaje automático** en el que la función en cascada se entrena a partir de muchas imágenes positivas y negativas. Luego se utiliza para detectar objetos en otras imágenes en tiempo real, haciendo uso de una función matemática (Wavelet Haar) propuesta por Alfred Haar en 1909.

Clasificador de Cascada

¿Que hace?

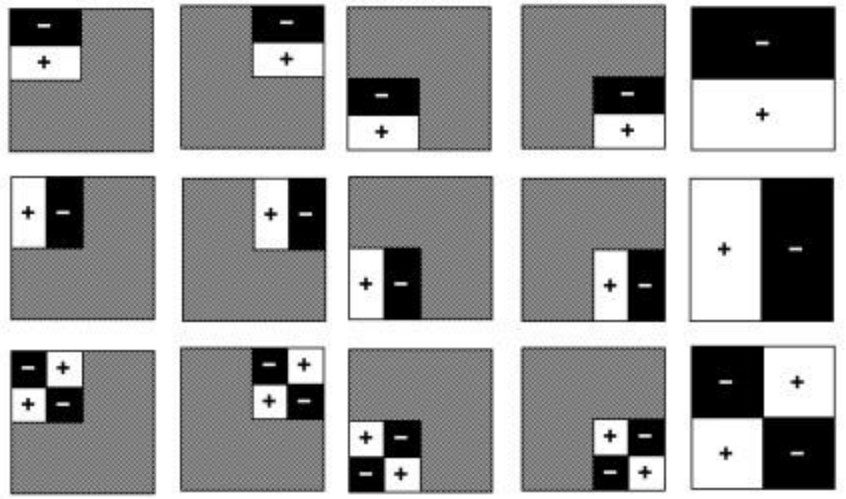
Definen regiones rectangulares sobre una imagen en escala de grises (imagen integral) y al estar formada por un numero finito de rectángulos, se puede obtener un valor escalar que consiste en sumar los pixeles de cada rectángulo, en base a una serie de clasificadores en cascada.

"Cada clasificador determina si la subregion se trata del objeto buscado o no"



Clasificadores Haar

Cálculo de haar sobre imágen



Áreas de búsqueda en la imagen

hacia lo alto!

Aplicar clasificador_Haar de caras frontales

1. Habilitar Google Drive
2. Importar librerías necesarias (opencv, numpy)
3. Cargar clasificador haar
4. Cargar imagen desde Google Drive y Aplicar filtro gris (COLOR_BGR2GRAY)
5. Identificar rostros en la imagen (detectMultiScale)
6. Dibujamos rectángulos de colores correspondientes a la matriz

Aplicar clasificador_Haar de caras frontales

1. Habilitar Google drive

```
from google.colab import drive
drive.mount('/content/drive')
```

2. Importar librerías

```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
```

Aplicar clasificador_Haar de caras frontales

Solo estamos detectando la cara frontal por lo tanto usaremos el archivo **haarcascade_frontalface_default.xml**

Este archivo permite usar los clasificadores en cascada basados en la función Haar para detectar una **cara frontal.**

3. Cargar clasificador Haar

```
face_cascade= cv2.CascadeClassifier('/content/drive/My Drive/…/haarcascade_frontalface_default.xml');
```

El archivo xml fue tomado de internet (https://github.com/opencv/opencv/tree/master/data/haarcascades)
Se encuentra ubicado en la carpeta de en drive
\USTA-202001\USTA-202001_7°_DEEP_LEARNING\Computer_vision\Python_files

4. Cargar imagen desde Google Drive

```
img = cv2.imread("/content/drive/…/images/caras.jpg")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

Aplicar clasificador_Haar de caras frontales

5. Identificar rostros en la imagen (detectMultiScale)

```
faces = face_cascade.detectMultiScale(gray,scaleFactor=1.1,minNeighbors=5,minSize=(30,30))
print(faces)
```

```
[[252  42  73  73]
 [  2  54  69  69]
 [150  42  96  96]
 [320  42  85  85]
 [ 76  54  89  89]
 [397  61  85  85]
 [474  78  87  87]
 [569  84  69  69]]
```

Matriz, donde cada fila es un cuadrado (x1,y1, x2, y2) donde se identifico una cara

Donde los parámetros son:

- **imagen** : matriz del tipo CV_8U que contiene una imagen donde se detectan objetos.
- **scaleFactor** : parámetro que especifica cuánto se reduce el tamaño de la imagen en cada escala de imagen (usamos 1.3, expandimos la imagen en un 30%)

- **minNeighbours** : Parámetro que especifica cuántos vecinos debe tener cada rectángulo candidato para retenerlo. Este parámetro afectará la calidad de las caras detectadas: un valor más alto produce menos detecciones pero con mayor calidad.

- **(optional)minSize** : Tamaño mínimo posible del objeto. Los objetos más pequeños que eso se ignoran.

Aplicar clasificador_Haar de caras frontales

6. Dibujamos rectángulos de color verde tomando como base las coordenadas guardadas en la matriz faces

```
[[252  42  73  73]
 [  2  54  69  69]
 [150  42  96  96]
 [320  42  85  85]
 [ 76  54  89  89]
 [397  61  85  85]
 [474  78  87  87]
 [569  84  69  69]]
```

```
for (x,y,w,h) in faces:
    cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,0),2)
cv2_imshow(img)
```

Aplicar clasificador_Haar de caras frontales

7. Código completo y mejorado (que nos enumere las caras)

```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow

face_cascade = cv2.CascadeClassifier('/content/…/haarcascade_frontalface_default.xml');
img = cv2.imread("/content/…/caras.jpg")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
faces = face_cascade.detectMultiScale(gray,1.3,2);
print(faces)
idx=0
for (x,y,w,h) in faces:
  cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,0),2)
  idx += 1
  cv2.putText(img,"Cara #{}".format(idx),(x,y-10),
  cv2.FONT_HERSHEY_SIMPLEX,.5,(0,255,0),1)
cv2_imshow(img)
cv2.imwrite('/content/…/caras_detectadas.jpg',img)
```

Realizar detecciones en un video o una cámara web

clasificador_Haar en un video o cámara web

1. Cargar video de Google drive
2. Características del video
2. Capturar cada frame del video
3. Aplicar filtro de gris al frame
4. Aplicar clasificador en el frame
5. dibujar rectángulo sobre el frame del video.
6. Grabar en un nuevo video el resultado

clasificador_Haar en un video o cámara web

0. ver video en Google colaboratory

```
from IPython.display import HTML
from base64 import b64encode
mp4 = open('/content/drive/My Drive/IA/Computer_Vision/Images/familia_cantando.mp4','rb').read()
data_url = "data:video/mp4;base64," + b64encode(mp4).decode()
HTML("""
<video width=400 controls>
      <source src="%s" type="video/mp4">
</video>
""" % data_url)
```

1. Cargar video de Google drive con opencv

```
videoentrada = cv2.VideoCapture('/content/drive/My Drive/…/images/familia_cantando.mp4')
```

2. Conociendo sus características

```
property_id = int(cv2.CAP_PROP_FRAME_COUNT)
totalframes = int(cv2.VideoCapture.get(videoentrada, property_id))
print( "total de frames: "+str(totalframes) )
print(str(videoentrada.get(3))+"x"+str(videoentrada.get(4))+" pixeles")
```

clasificador_Haar en un video o cámara web

2.1. extrayendo solo unos frames

```
#recorremos los 6 primeros frames de 1412
for x in range(6):
  videoentrada.set(1,x);
  #ret: retorno (true si es frame se lee correctamente)
  ret, frame = videoentrada.read()
  cv2_imshow(frame)
```

2.2. Recorriendo todo el video

```
while(videoentrada.isOpened()):
  ret, frame = videoentrada.read()
  if ret == True:
    # Nuestras operaciones sobre los frames se hacen aqui
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
  else:
    break
videoentrada.release()
```

clasificador_Haar en un video o cámara web

3. Cargamos el clasificador haar

```
face_cascade = cv2.CascadeClassifier('/content/…/Python_files/HaarCascade/haarcascade_frontalface_default.xml')
```

4. Aplicar filtro gris a cada frame del video

```
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

5. Aplicamos el clasificador haar a la imagen gris

```
faces = face_cascade.detectMultiScale(gray,scaleFactor=1.1,minNeighbors=5)
```

6. Dibujamos rectángulos en los lugares que se identificaron caras

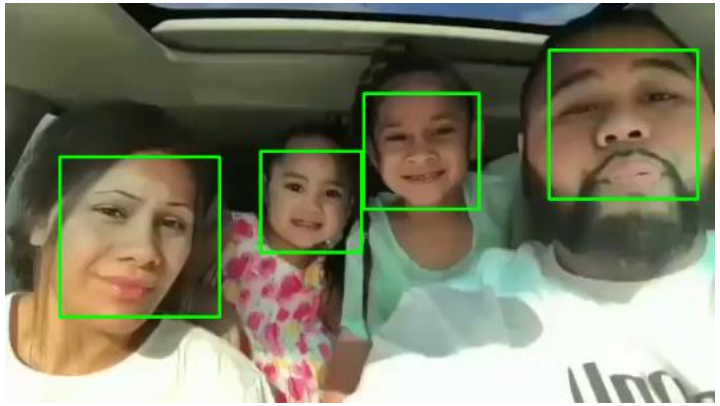
```
for (x, y, w, h) in faces:
    cv2.rectangle(frame,(x,y),(x+w, y+h),(0, 255, 0),2)
cv2_imshow(frame)
```

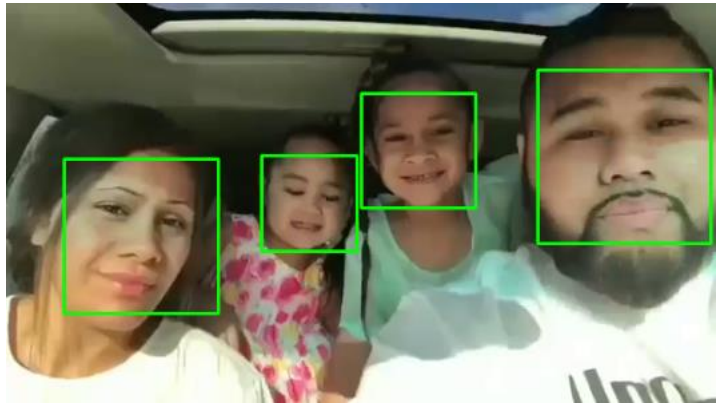
clasificador_Haar en un video o cámara web

7. Aplicando todo

```
#video que vamos a analizar
video = cv2.VideoCapture('/content/…/familia_cantando.mp4')
#clasificador haar
face_cascade = cv2.CascadeClassifier('/content/…/haarcascade_frontalface_default.xml')

for x in range(6):
  videoentrada.set(1,x);
  #ret: retorno (true si es frame se lee correctamente)
  ret, frame = videoentrada.read()
  #convertimos el frame a gris
  gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
  #detectamos las caras en el frame
  faces = face_cascade.detectMultiScale(gray,scaleFactor=1.1,minNeighbors=5)
  for (x, y, w, h) in faces:
    #dibujamos rectángulos verdes alrededor de las caras
    cv2.rectangle(frame,(x,y),(x+w, y+h),(0, 255, 0),2)
  cv2_imshow(frame)
```





Exportar video con caras

Instalar libreria para medir la ejecución

```
# instalar librería para visualizar el progreso de ejecución una tarea en background
pip install pyprind
```

```
import sys
import time          #  calcular tiempo (en este caso tiempo de descarga de archivo)
import pyprind
#funcion para ver el avance de procesos en background
def reporthook(step, total_step):
    global start_time
    if step == 0:
        start_time = time.time()
        return
    duration = time.time() - start_time
    speed = total_step / (1024.**2 * duration)
    percent = step *100 /total_step
    sys.stdout.write("\r%d pasos | %d frames -> %%.2f frames | %d segundos transcurrido" %
                     (step, total_step, percent, duration))
    sys.stdout.flush()
```

Exportar video con caras

```
face_cascade = cv2.CascadeClassifier('/content/drive/My
Drive/IA/Computer_Vision/Python_files/HaarCascade/haarcascade_frontalface_default.xml')
#video que analizaremos
videoentrada = cv2.VideoCapture('/content/drive/My Drive/IA/Computer_Vision/Images/familia_cantando.mp4')
#video resultante del análisis, definimos el codec DIVX
codec = cv2.VideoWriter_fourcc(*'DIVX')
# Definimos el fps = 20.0 y el tamaño de cada frame (640x360)
videosalida = cv2.VideoWriter('/content/drive/My Drive/IA/Computer_Vision/Images/familia_cantando_haar2.avi',
                              codec, 20.0,(640,360))
framenum=0
while (videoentrada.isOpened()):
  ret, frame = videoentrada.read()
  if ret == True:
    gray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray,scaleFactor=1.1,minNeighbors=5)
    for (x,y,w,h) in faces:
      cv2.rectangle(frame,(x,y),(x+w,y+h),(0,255,0),2)
    videosalida.write(frame)
    reporthook(framenum, 1412)
    framenum=framenum+1
  else: break
videoentrada.release()
videosalida.release()
```

Talleres

Taller 1:

Implementar un **detector** de ojos y sonrisas

Taller 2:

Implementar un **detector** de cuerpos humanos

4. Creando un modelo para reconocer rostros personales

https://www.teknotut.com/en/facial-recognition-with-raspberry-pi-and-opencv/

UNIVERSIDAD SANTO TOMÁS

PRIMER CLAUSTRO UNIVERSITARIO DE COLOMBIA

SECCIONAL TUNJA

VIGILADA MINEDUCACIÓN - SNIES 1732

¡Siempre hacia lo alto!

USTATUNJA.EDU.CO

@santotomastunja