

Tarea 10. Seguimiento de Rostros método tradicional vs método Deep Learning

Dr. Francisco Javier Hernández
Alumno: Mario Alberto Tapia de la Cruz

Maestría en Matemáticas Aplicadas

0.1. Método tradicional

Para esta tarea se usarán algunas funciones de la librería OpenCV2 para implementar un programa que haga seguimiento de rostros en Python. A continuación se mencionarán las funciones a utilizar.

- **cv2.CascadeClassifier** - Con esta función cargaremos un clasificador entrenado para hacer la detección de rostros, usaremos un clasificador Haar.
- **cv2.data.haarcascades** - Esto va como argumento en la función **CascadeClassifier**, nos ayudará a cargar nuestro archivo entrenado .xml sin necesidad de escribir rutas de forma manual.
- **detectMultiScale** - Esta función escanea nuestra imagen en múltiples tamaños y nos ayudará a detectar objetos (en nuestro caso, rostros), y como resultado nos generará una lista de tuplas (x, y, w, h) donde se hicieron las detecciones; cada una representa un rectángulo. A la función se le dará de entrada los siguientes parámetros:
 - **img_grises** - Una imagen en escala de grises.
 - **scaleFactor** - Cuánto reduce el tamaño de la ventana de detección de escala.
 - **minNeighbors** - Vecinos que necesita una región candidata para ser aceptada como rostro.
 - **minSize** - Tamaño mínimo en píxeles que puede tener una cara.
- **cv2.matchTemplate** - Esta función compara la imagen con el template obtenido, y usaremos el método **TM_CCOEFF_NORMED** para generar un mapa de correlación normalizada.
- **cv2.minMaxLoc** - Esta función la usaremos para encontrar el mínimo y el máximo de la matriz generada por **matchTemplate**, además de las coordenadas donde ocurren dichos valores.

Se probará con cuatro videos:

- prueba2.mp4 (mismo de la tarea pasada, se referirá a él como video_Tapia)
- walking_london.mp4
- video_walking_nuevo_hd.mp4
- video_dancing.mp4

Para la detección de rostros cargamos el clasificador Haar pre-entrenado **haarcascade_frontalface_default.xml**. Este clasificador fue entrenado para detectar rostros frontales. Y también se probará con el clasificador Haar pre-entrenado para detectar ojos de forma frontal, **haarcascade_eye_tree_eyeglasses.xml**.

Además para contabilizar rostros únicos utilizamos dos filtrados:

- Intersection over Union (IoU): es el cociente entre las intersecciones de áreas y las uniones de áreas entre ambas bounding boxes de dos rostros.
- Distancias de centros de bounding boxes, además si IoU está por encima de un umbral dado (0.3), pasa a comparar las distancias de los centros de las cajas para verificar que se trata de una cara nueva.

0.2. ¿Cómo funciona el programa?

El pipeline es bastante similar al de la tarea pasada, a excepción de que ahora está detectando varios rostros por frame y dibuja la bounding box respectiva para el rostro detectado, además de que se utilizan las dos funciones para filtrar rostros distintos mencionadas anteriormente. Para la comparación de rostros, compara el rostro actual detectado con los rostros guardados en la lista `detected_faces`, de ser un rostro que no estaba ahí, lo agrega a la lista de rostros nuevos y suma uno al contador.

A continuación se presentan algunas capturas de pantalla de varios rostros detectados.

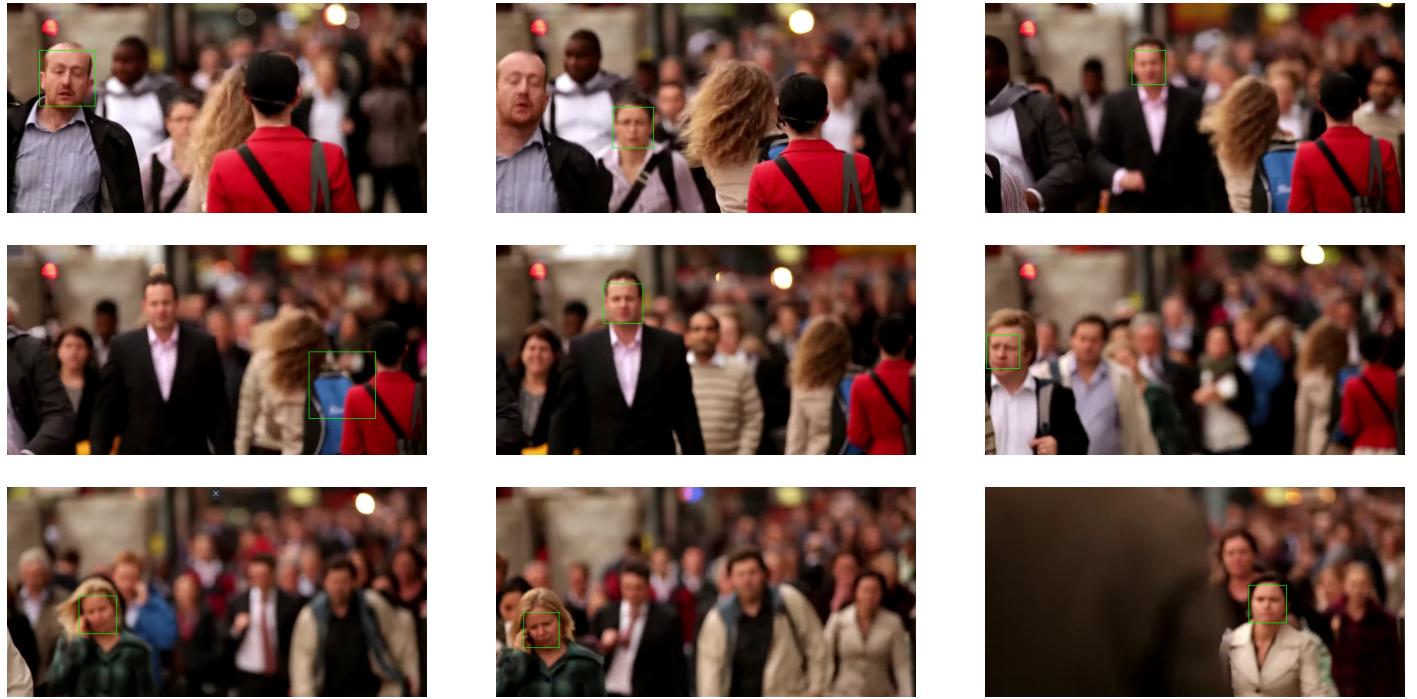


Figura 1: **Rostros Detectados 9, Video Walking London.** Parámetros: scaleFactor = 1.1, minNeighbors=8, minSize=(100,100)



Figura 2: **Rostros Detectados 10, Video Dancing.** Parámetros: $\text{scaleFactor} = 1.1$, $\text{minNeighbors}=8$, $\text{minSize}=(100,100)$



Figura 3: **Rostros Detectados Video Tapia.** Parámetros: scaleFactor = 1.1, minNeighbors=8, minSize=(100,100)

0.3. Método en Deep Learning

Para esta tarea se usarán algunas funciones de la librería OpenCV2 para implementar un programa que haga seguimiento de rostros en Python. A continuación se mencionarán las funciones a utilizar.

- MTCNN (Multi-Task Cascaded Convolutional Networks - Para la detección de rostros, un método importado de la librería `facenet_pytorch`.
- Seguimiento usando DeepSORT (Deep Simple Online and Realtime Tracking) - Importado de la librería `deep_sort_realtime`

El conteo de `unique_ids` viene directamente del ID que asigna DeepSORT, por lo que para el conteo de rostros simplemente se cuenta `len(unique_ids)`. Todo esto se lleva a cabo con el uso de `tracker.update_tracks(detections, frame=frame)`. En las variables

- tracker - Guardamos el método, que inicializamos con treinta épocas
- tracks - Nos lleva el seguimiento de cada ID y sus características.

A continuación se mostrarán ejemplos de las escenas Walking London y Tapia.

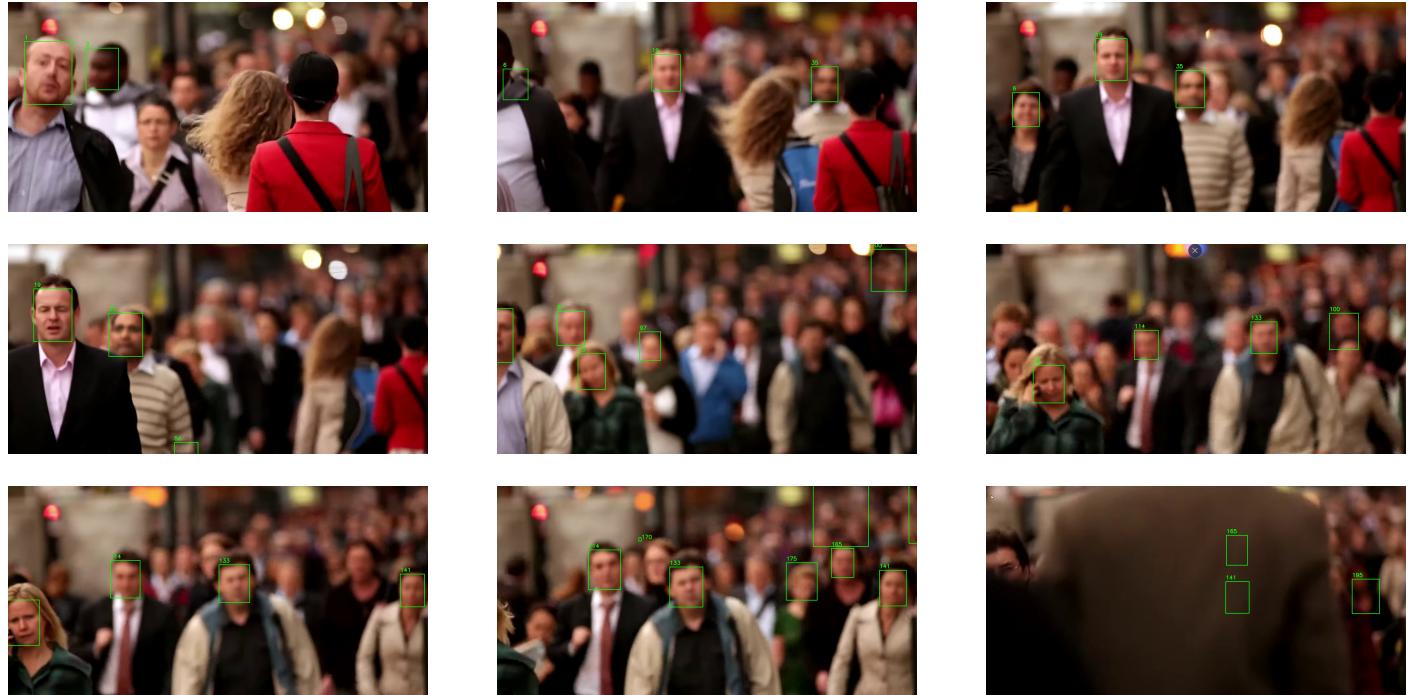


Figura 4: Walking London, 45 People, DeepSORT Implementation

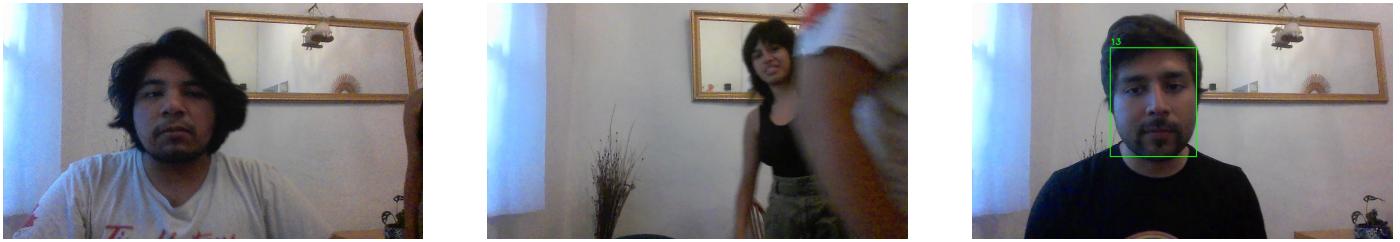


Figura 5: Tapia, 4 People Detected (3 in Scene), DeepSORT Implementation

0.4. Conclusiones

El conteo a mano fue únicamente de las caras que se veían completamente de frente (al menos para el vídeo de Tapia, walking HD dancing), para walking london se tomaron las caras que se podían distinguir en la escena (al ser una escena muy pixeleada), aunque se puede apreciar que el método de DeepSORT pudo detectar más caras, aquellas que apenas se podían empezar a notar en la escena en muchas de las ocasiones. No se dejarán capturas de la corrida del código con el video de walking HD, pero el vídeo se adjuntará en la carpeta junto con los demás videos. Se dejará una tabla comparativa a continuación.

	A mano	SF = 1.1, mN=8, mS=100	SF = 1.1, mN=4, mS=40	DL
walking HD	16	9	13	No se probó
walking London	12	9	9	45
dancing	10	10	17	No se probó
Tapia	3	2	1	4

Cuadro 1: Tabla comparativa: Conteo a mano vs Parámetros 1 vs Parámetros 2