Sistema para el conteo automático de personas utilizando YOLO con OpenCV

MANUAL DEL USUARIO

Versión 1

Karen Isabel Torres
Erika Vásquez Tapia

Introducción

Durante varias décadas los científicos han perseguido la construcción de algoritmos capaces de procesar información al igual que el cerebro humano. Alan Turing fue el primero en estudiar el cerebro como una forma de ver el mundo de la computación. Sin embargo, los primeros teóricos que concibieron los fundamentos de la computación neuronal fueron Warren McCulloch y Walter Pitts, quienes en 1943 lanzaron una teoría acerca de la forma de trabajar de las neuronas.

Las redes neuronales fueron originalmente una simulación abstracta de los sistemas nerviosos biológicos, constituidos por un conjunto de unidades llamadas neuronas o nodos conectados unos con otros. El primer modelo de red neuronal fue propuesto en 1943 por McCulloch y Pitts en términos de un modelo computacional de actividad nerviosa. Este era un modelo binario, donde cada neurona tenía un escalón o umbral prefijado, y sirvió de base para los modelos posteriores.

Con el pasar del tiempo el desarrollo de métodos capaces de detectar objetos mediante inteligencia artificial ha ido evolucionando, y se ha hecho realidad en la actualidad gracias al avance tecnológico en cuanto al procesamiento de la información. El presente trabajo muestra una de las técnicas más utilizadas para la detección de objetos, denominada Yolo que se basan en el uso de redes neuronales y métodos descriptores y clasificadores.

Sección 1: Introducción general de herramientas

Python

Python destaca por su facilidad de uso, legibilidad, portabilidad y simplicidad. Además, este lenguaje de programación suele ser el predilecto para realizar temas de inteligencia artificial debido a varios factores:

- Una gran comunidad
- Una mayor cantidad de bibliotecas si lo comparamos con otros lenguajes, hay una gran cantidad de recursos open-source para trabajar con temas de IA.
- Los prototipos se pueden programar más rápido, al ser python un lenguaje dinámico y con un tipado débil, esto permite desarrollar a gran velocidad.

OpenCV

Open CV (en sus siglas en inglés, Open Source Computer Vision) es una librería de programación de código abierto dirigida a la visión por computador, desarrollada por Intel. Es una librería multiplataforma, existiendo versiones para GNu/Linux, Mac OS, Windows y Android. Se ha utilizado en varias aplicaciones, desde sistemas de seguridad con detección de movimiento, hasta aplicaciones de control de procesos donde se requiere reconocimiento de objetos.

Open CV contiene más de 500 funciones que abarcan una gran gama de áreas en el proceso de visión, como reconocimiento de objetos (reconocimiento facial), calibración de cámaras, visión estéreo, visión robótica, clasificar acciones en video, convertir imágenes, extraer modelos 3D, crear espacio 3D desde una imagen de cámaras estéreo creando imágenes de alta calidad mediante la combinación de imágenes de baja calidad (Caballero, 2017).

Anaconda

Anaconda es una distribución libre y abierta de los lenguajes Python y R, utilizada en ciencia de datos, y aprendizaje automático. Incluye procesamiento de grandes volúmenes de información, análisis predictivo y cómputos científicos. Está orientado a simplificar el despliegue y administración de los paquetes de software. Tiene la mayoría de las bibliotecas de Aprendizaje automático y Aprendizaje profundo y es fácil de usar y es un Cuaderno interactivo de Python (Gutiérrez & Galar, 2019).

Tensorflow

Es una biblioteca de código abierto para aprendizaje automático a través de un rango de tareas, y desarrollado por Google para satisfacer sus necesidades de sistemas capaces de construir y entrenar redes neuronales para detectar y descifrar patrones y correlaciones, análogos al aprendizaje y razonamiento usados por los humanos. Cuenta con un API para la detección de objetos así como también para el conteo, la cual soporta modelos de redes neuronales ya entrenados en extensiones .pb (Farias & Guerrero, 2019) .

NumPy

En el mundo de Python, las matrices NumPy son la representación estándar de datos numéricos y permiten la implementación eficiente de cálculos numéricos en un lenguaje de alto nivel. NumPy se puede mejorar a través de tres técnicas: vectorización cálculos, evitando copiar datos en la memoria y minimizando los recuentos de operaciones.

Darflow

Es la implementación “oficial” de YOLO, creada por las mismas personas detrás del algoritmo. Está escrito en C con CUDA, así que soporta cómputo con GPU. En realidad, es todo un framework para redes neuronales, así que se puede usar para otros objetivos además de YOLO.

Sección 2: Pasos para la instalación

Instalación de YOLO

Requerimientos:

- Anaconda Navigator
- Python 3.5 or 3.6
- Tensorflow
- OpenCV3
- Darkflow
- Numpy
- Cython
- Visual Studio https://visualstudio.microsoft.com/es/downloads/

1. Instalar ambiente Anaconda

Necesitarás tener listo tu ambiente de desarrollo local, en tu computadora de escritorio o portátil.

En este paso veremos como descargar anaconda a nuestro disco y obtener esta suite científica de Python
Nos dirigimos a la página de Anaconda y luego a la sección de Download (descargas)
Elegimos nuestra plataforma: Windows, Mac o Linux (en mi caso seleccionaré la de Apple)

Figura 1. *Instalador de Anaconda*

Atención: Elegir la versión de Python 3.6 (y no la de 2.7) y seleccionar el instalador Gráfico (Graphical Installer)

Anaconda viene con una suite de herramientas gráficas llamada “Anaconda Navigator”.

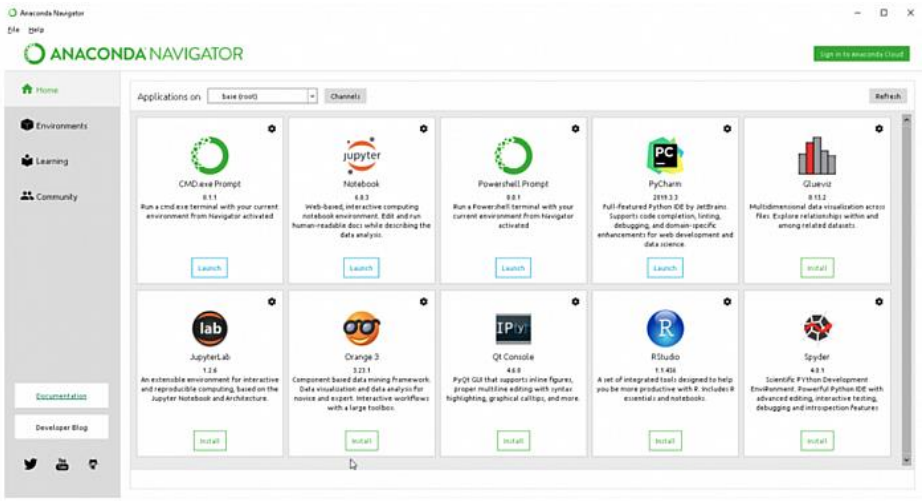


Figura 2. *Anaconda Navigator*

2. Instalar de librerías

- TensorFlow: *pip install tensorflow*
- *OpenCV: pip install opencv*

3. Desacargar Darkflow del repositorio

Darknet es un marco de red neuronal de código abierto escrito en C y CUDA. Es rápido, fácil de instalar y admite cómputo de CPU y GPU.
En el siguiente repositorio se encuentra el archivo darkflow:
https://github.com/llSourcell/YOLO_Object_Detection
Se debe construir la siguiente librería para la instalación de darkflow desde el cmd python *setup.py build_ext –inplace o pip install -e*

4. Descargar los pesos a utilizar

A continuación, descargamos los pesos pre-entrenado (entrenamiento oficial) y guardaremos en el directorio “bin” contenido en darkflow.
Se descargó los pesos YOLOv2 608x608 de la siguiente página: https://pjreddie.com/darknet/yolov2/

Para comprobar si la instalación de Yolo se realizó correctamente, ingrese el siguiente código para probar su funcionamiento con las imágenes de muestra:

python flow --model cfg/yolo.cfg --load bin/yolov2.weights --imgdir sample_img/

5. Procesar el video

Para ejecutar el video se ejecuta el siguiente comando:

python camara.py

Comando para obtener el video resultante de la detección

python flow --model cfg/yolo.cfg --load bin/yolov2.weights --demo videoA3.mp4 --gpu 1.0 --saveVideo

Sección 3: Ejecución

- **Entrada**

Para el desarrollo del proyecto se tiene como entrada tres videos donde se encuentran varias personas que son los objetos que se van a detectar.
Dentro del código python donde se programa la detección de personas se utiliza la libreria cv2, esta librería nos ayudara a obtener la entrada del video por cada frame utilizando comando ***VideoCapture*** tal como se observa en la Figura 1.

```
capture = cv2.VideoCapture('C:/Users/karee/anaconda3/darkflow/personas.mp4')
colors = [tuple(255 * np.random.rand(3)) for i in range(5)]
time.sleep(0.02)
```

Figura 3: *Código en Python para capturar el video*

- **Proceso**

De acuerdo con la arquitectura planteada, misma que se puede observar en el Anexo 1. Se plantea trabajar con el algoritmo Yolo, este algoritmo maneja la detección de varios objetos, por tal motivo se limitó a trabajar solo con la clase ***Person*** con la finalidad de reducir tiempos de procesamiento y que la detección de personas sea más precisa. A continuación, en la Figura 2 se puede observar el código que se ha utilizado para realizar la detección y el conteo.

Internamente el algoritmo realiza la captura de frame por un determinado tiempo, por cada frame capturado realiza la detección del objeto e inmediatamente lo cuenta para ser almacenado en un archivo txt.

Se ha comenzado con la captura del video y obtener los frame por segundo. La condición que se refleja en el código es para enmarcar los objetos detectados en un cuadro, este cuadro se lo pudo realizar obteniendo los puntos de inicio - fin de X y Y, que además sirvieron para obtener el punto medio que fue utilizado para realizar el conteo de los objetos detectados que pases por un umbral previamente definido.

```python
while (capture.isOpened()):
    stime = time.time()
    ret, frame = capture.read()
    if ret:
        results = tfnet.return_predict(frame)
        for color, result in zip(colors, results):
            confidence = (result['confidence'])
            tl = (result['topleft']['x'], result['topleft']['y'])
            br = (result['bottomright']['x'], result['bottomright']['y'])
            label = result['label']
            text = '{}: {:.0f}%'.format(label, confidence * 100)
            frame = cv2.rectangle(frame, tl, br, color, 7)
            frame = cv2.putText(frame, text, tl, cv2.FONT_HERSHEY_COMPLEX, 1, (0, 0, 0), 2)
            cv2.line(frame,(0,150),(800,150),(0,0,255),2)
            #punto medio
            inicioX = (result['topleft']['x'])
            inicioY = (result['topleft']['y'])
            finX = (result['bottomright']['x'])
            finY = (result['bottomright']['y'])
            puntoMedioX = (inicioX+finX)/2
            puntoMedioY = (inicioY+finY)/2
            x = int(puntoMedioX)
            y = int(puntoMedioY)
            #circulo
            cv2.circle (frame, (x,y),1,(0,0,255), 2)
```

Figura 4: *Código en Python*

- **Salida**

Por defecto, YOLO solo muestra objetos detectados con una confianza de .25 o superior. Como se puede observar en la siguiente Figura 3, al ejecutar el codigo py se obtenido como resultado la detección de cada objeto con el porcentaje de confianza y el tipo de objeto.

Figura 5: *Conteo de personas*

Finalmente, los datos del conteo serán almacenados en un archivo de texto para realizar un análisis de los resultados obtenidos, y con ello identificar la precisión del algoritmo utilizado.

Sección 4: Resultados de ejecución y generación de reportes

Para comprobar la precisión del algoritmo YOLO, se ha tomado como muestra 3 videos que simulan la entrada y salida de personas de un determinado sitio.

Como se puede observar en la Tabla 1, se refleja el conteo manual y automático de cada video procesado, tomando en cuenta si estos están entrando o saliendo, la última columna de la tabla presenta la precisión obtenida utilizando el algoritmo YOLO.

Tabla 1. Resultados obtenidos de la detección en tiempo real

Videos	Conteo Manual		Conteo Automático		Precisión
	Entrada	Salida	Entrada	Salida	%
Video 1	3	7	3	6	**90%**
Video 2	3	4	2	3	**72%**
Video 3	3	5	1	5	**75%**

Video 1

Implementando el algoritmo de Yolo para el conteo de personas en el video 1 se obtuvo una precisión del 90%.

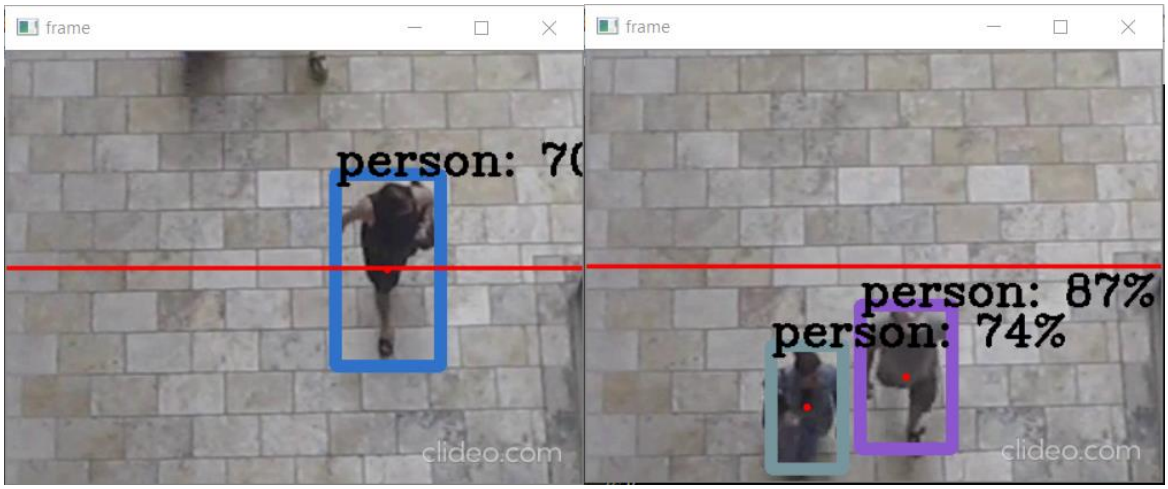


Figura 5: *Video resultante de la detección y conteo*

El conteo de personas por "entrada" fue excelente ya que el conteo manual como automático coincide, por otro lado, el conteo de personas por "salida", no detectó una persona ya que al trabajar en una máquina local que no cuenta con una GPU la detección es un lenta, ocasionando problemas en el conteo pero la detección es certera.

Video 2

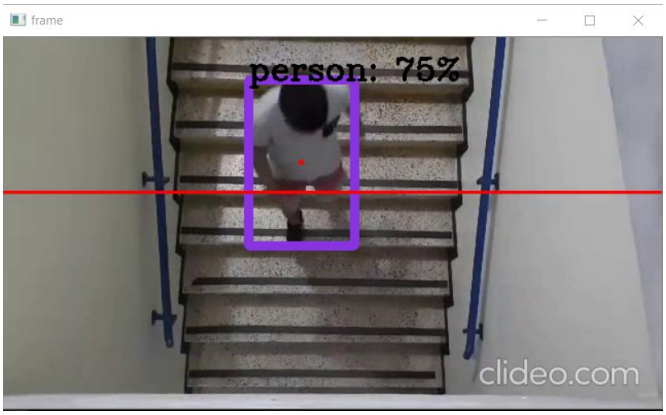


Figura 6: *Video resultante de la detección y conteo*

En el video 2 se obtuvo una precisión del 72%, al igual que el video 1 la detección fue eficiente, pero el conteo débil debido al lento tiempo de detección al implementar el algoritmo en un computador sin GPU.

Video 3

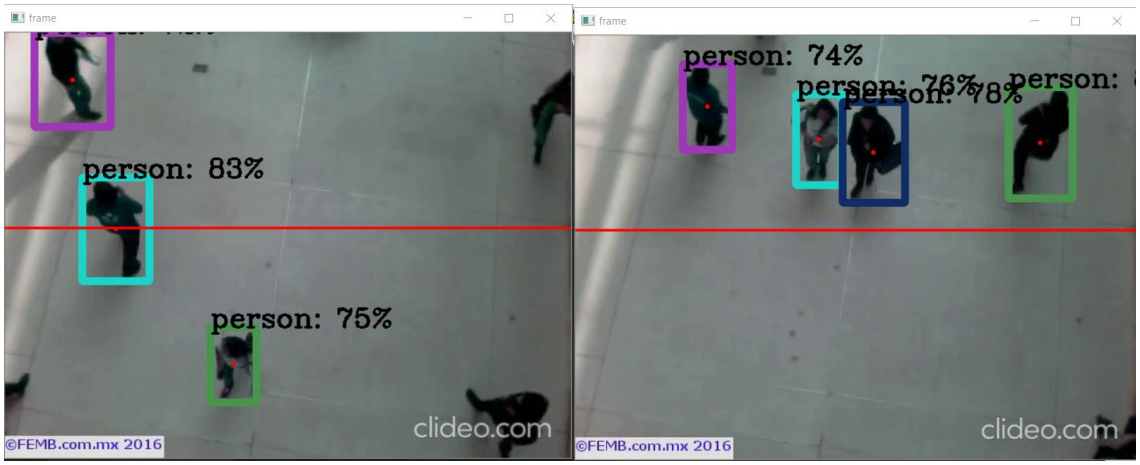


Figura 7: *Video resultante de la detección y conteo*

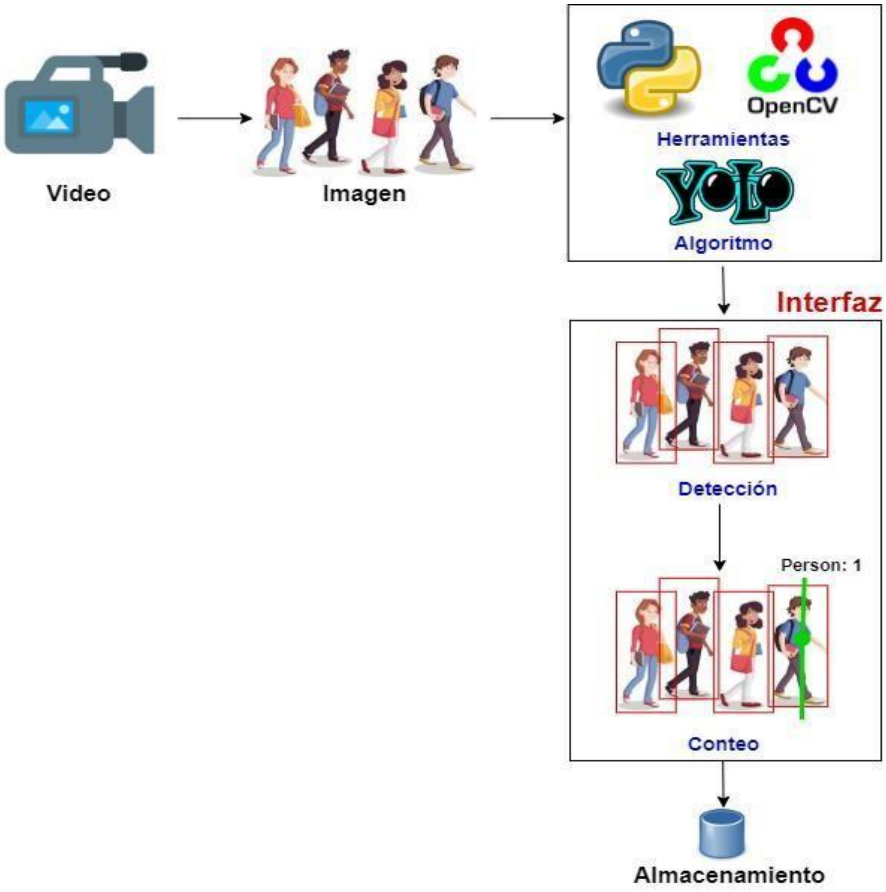
Al implementar el código con el video 3 se obtuvo una precisión del 75%, como se puede observar en la Tabla 1 la precisión de conteo con los objetos de entrada fue débil con una desventaja de 3 a 1 comparando el conteo manual con el automático. A diferencia de los objetos de salida cuya precisión fue exacta. Cabe recalcar que la falta de precisión fue por la lenta detección de los objetos.

Conclusiones

- El algoritmo Yolo es una muy buena alternativa para la detección de objetos ya que cuenta con unas 80 clases de objetos que pueden ser utilizados para cualquier fin.
- Yolo funciona con más eficiencia con equipos o entornos que cuenten con características de procesamiento avanzadas como GPU, etc.
- Al definir los umbrales para el conteo es necesario tomar en cuenta el rango que se va a considerar, ya que si dicho rango es muy elevado el conteo podría duplicarse o de lo contrario al ser el rango muy pequeño no se contaría el objeto.

- Trabajar con entornos en la nube es muy recomendado ya que el tiempo de procesamiento para la detección de objetos se reduce considerablemente.

ANEXOS



Anexo 1. Arquitectura de la Solución