

VISION ARTIFICIAL

Implementacion IA para reconocimiento y localización de objetos en una imagen y seguimiento (versión 1).

Objetivos

El presente documento pretende detallar como aplicar un algoritmo, basado en aprendizaje profundo. llamado YOLOv3 (You Only Look Once) capaz de detectar y posicionar un objeto en concreto en una imagen o frame y por extensión en un video. Además, pretendemos hacer un seguimiento de cada objeto frame a frame lo que quiere decir que debemos identificarlo en todo momento (por ejemplo al detectar un coche le asignaríamos un número único y que no lo perdiera hasta sacarlo del plano del video) y para ello utilizaremos otro algoritmo llamado SORT (Simple Online y Realtime Tracking) para hacer dicho seguimiento.

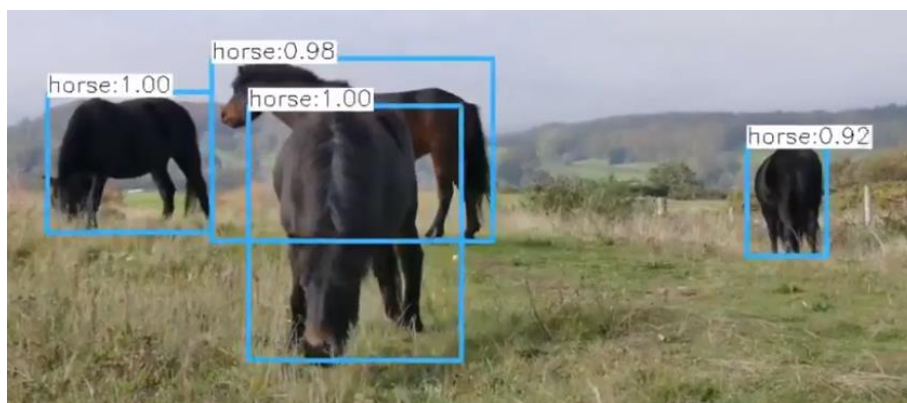


Fig1. Ejemplo fotografía y resultados con YOLOv3

1	person
2	bicycle
3	car
4	motorbike
5	aeroplane
6	bus
7	train
8	truck
9	boat
10	traffic light
11	fire hydrant
12	stop sign
13	parking meter
14	bench
15	bird
16	cat
17	dog
18	horse
19	sheep
20	cow
21	elephant
22	bear
23	zebra
24	giraffe
25	backpack
26	umbrella

Fig2. Lista objetos

Como se puede apreciar en la imagen, **Fig.1**, **YOLOv3** forma un recuadro alrededor de la imagen que reconoce, posicionandolo y te dice qué objeto ha reconocido además del porcentaje de que sea el objeto en cuestión.

Esta última versión, de **YOLOv3**, puede reconocer hasta 80 objetos diferentes (**Fig.2** coches, mesas, pájaros.etc). El algoritmo en cuestión, lo hemos utilizado por su gran rapidez y su gran acierto.

Probando YOLOv3

Lo primero que haremos será probar **YOLOv3** y que potencial tiene y si se adapta a nuestro objetivo.

Para trabajar usaremos **Linux** como sistema operativo. Si es usuario de **windows 10**, por ejemplo, puede descargar desde **Microsoft** una versión de **Linux** totalmente compatible y la forma sencilla de como instalarla en este enlace:

<https://docs.microsoft.com/es-es/windows/wsl/install-win10>

Además, como lenguaje utilizaremos **python3**. Podemos instalarlo si no lo tenemos ejecutando en el terminal el siguiente código. Antes actualizaremos los paquetes, en este caso Ubuntu:

```
$ sudo apt update
```

```
$ sudo apt install python3
```

Ahora instalaremos **opencv** que es una librería libre de visión artificial para interactuar con las imágenes y que usaremos por ejemplo, para dibujar el cuadrado alrededor del objeto que detectamos:

```
$ sudo apt install open-cv
```

Ahora comprobaremos que la instalación ha sido correcta, abriendo el **python3** en la consola:

```
$ python3
```

Escribiendo una vez abierto:

```
>>> import cv2
```

```
>>> cv2.__version__
```

Devolviéndonos la versión instalada de **opencv**: '3.4.4'

Si vamos a **opencv** para consultar la última versión para **python** veremos que seguramente no la tenemos actualizada: <https://opencv.org/releases/>

Así que lo que vamos a hacer es instalar un gestor de paquetes de **python** llamado **pip** para instalar la última versión de **opencv**.

```
$ sudo apt install python3-pip
```

Una vez instalado el gestor de paquetes, instalaremos la última versión de **opencv** escribiendo:

```
$ pip3 install opencv-python
```

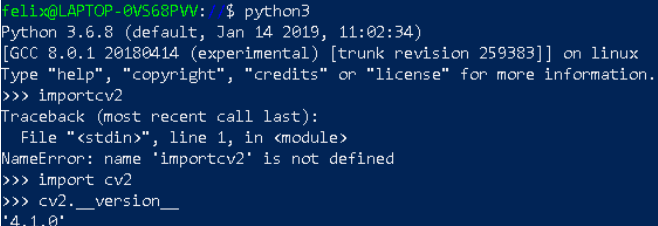
Una vez hecho esto, consultaremos la nueva versión instalada como hicimos anteriormente desde **\$ python3**:

```
>>> import cv2
```

```
>>> cv2.__version__
```

Devolviéndonos la nueva versión (Fig.3)

instalada en mi caso de **opencv**: '4.1.0'



```
felix@LAPTOP-0VS68PVV: / $ python3
Python 3.6.8 (default, Jan 14 2019, 11:02:34)
[GCC 8.0.1 20180414 (experimental) [trunk revision 259383]] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> importcv2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'importcv2' is not defined
>>> import cv2
>>> cv2.__version__
'4.1.0'
```

Fig.3 Consola Ubuntu

Ahora descargaremos el siguiente proyecto de github:

<https://github.com/spmallick/learnopencv/tree/master/ObjectDetection-YOLO>

y procederemos a ejecutar los siguientes comandos donde hallamos descargado el proyecto, en home sería lo idea, como se describe en el.

```
sudo chmod a+x getModels.sh
```

```
./getModels.sh
```

Para detectar en la imagen o un video completo los 80 objetos (lista en **coco.names**) con lo que está entrenado **YOLOv3**, se ejecutaran diferentes instrucciones en la linea de comandos.

En el proyecto viene de base una foto llamada bird.jpg, que es la que se llama en la instrucción así como en el video llamado run.mp4. Si se quiere detectar objetos en una imagen o video propio, se ha de sustituir esos nombres y su extensión por los propios.

Imagen:

```
python3 object_detection_yolo.py --image=bird.jpg
```



Fig.4 bird_yolo_out_py.jpg

Video:

```
python3 object_detection_yolo.py --video=run.mp4
```

Por ejemplo, si queremos probar un video llamado juegodetronos.mp4 nuestro, la instrucción por consola sería así:

```
python3 object_detection_yolo.py --video=juegodetronos.mp4
```

Una vez ejecutada la instrucción el programa creará un video o imagen (Fig.4) con su nombre mas **_yolo_out_py.avi/.jpg**, en nuestro caso:

juegodetronos_yolo_out_py.avi

Código Python

```
# This code is written at BigVision LLC. It is based on the OpenCV project. It is subject to the
license terms in the LICENSE file found in this distribution and at
http://opencv.org/license.html
```

```
# Ejemplo: python3 object_detection_yolo.py --video=run.mp4
#          python3 object_detection_yolo.py --image=bird.jpg
```

```
import cv2 as cv
import argparse
import sys
import numpy as np
import os.path
```

```
# Initialize the parameters
confThreshold = 0.5 #Confidence threshold
nmsThreshold = 0.4 #Non-maximum suppression threshold
inpWidth = 416     #Width of network's input image
inpHeight = 416    #Height of network's input image
```

```
parser = argparse.ArgumentParser(description='Object Detection using YOLO in OPENCV')
parser.add_argument('--image', help='Path to image file.')
parser.add_argument('--video', help='Path to video file.')
args = parser.parse_args()
```

```
# Load names of classes
classesFile = "coco.names"
classes = None
with open(classesFile, 'rt') as f:
    classes = f.read().rstrip('\n').split('\n')
```

```
# Give the configuration and weight files for the model and load the network using them.
modelConfiguration = "yolov3.cfg"
modelWeights = "yolov3.weights"
```

```
net = cv.dnn.readNetFromDarknet(modelConfiguration, modelWeights)
#Demandar a la xarxa que utilitzi un fons de càlcul específic on sigui compatible.
net.setPreferableBackend(cv.dnn.DNN_BACKEND_OPENCV)
#seleccionamos CPU / Se podria hacer con cv.dnn.DNN_TARGET_OPENCL (GPU) pero solo si es IBM. En
teoria cambia si no es IBM a CPU.
net.setPreferableTarget(cv.dnn.DNN_TARGET_CPU)
```

```
# Get the names of the output layers
def getOutputsNames(net):
    # Get the names of all the layers in the network
    layersNames = net.getLayerNames()
    # Get the names of the output layers, i.e. the layers with unconnected outputs
    return [layersNames[i][0] - 1] for i in net.getUnconnectedOutLayers()
```

```
# Draw the predicted bounding box
def drawPred(classId, conf, left, top, right, bottom):
    # Draw a bounding box.
    cv.rectangle(img, (left, top), (right, bottom), (255, 178, 50), 2)
    '''
    Python: cv.Rectangle(img, pt1, pt2, color, thickness=1, lineType=8, shift=0) → None
    img - Image.
        pt1 - Vertex of the rectangle.
        pt2 - Vertex of the rectangle opposite to pt1 .
        rec - Alternative specification of the drawn rectangle.
        color - Rectangle color or brightness (grayscale image).
        thickness - Thickness of lines that make up the rectangle. Negative values, like
CV_FILLED , mean that the function has to draw a filled rectangle.
        lineType - Type of the line. See the line() description.
        shift - Number of fractional bits in the point coordinates.
    Grueso linea
    '''
```

```
label = '%.2f' % conf
```

```
# Get the label for the class name and its confidence
if classes:
```

```

        assert(classId < len(classes))
        label = '%s:%s' % (classes[classId], label)

        #Display the label at the top of the bounding box
        labelSize, baseLine = cv.getTextSize(label, cv.FONT_HERSHEY_SIMPLEX, 0.5, 1)
        top = max(top, labelSize[1])
        cv.rectangle(frame, (left, top - round(1.5*labelSize[1])), (left + round(1.5*labelSize[0]),
top + baseLine), (255, 255, 255), cv.FILLED)
        cv.putText(frame, label, (left, top), cv.FONT_HERSHEY_SIMPLEX, 0.75, (0,0,0), 1)

# Remove the bounding boxes with low confidence using non-maxima suppression
def postprocess(frame, outs):
    frameHeight = frame.shape[0]
    frameWidth = frame.shape[1]

    # Scan through all the bounding boxes output from the network and keep only the
    # ones with high confidence scores. Assign the box's class label as the class with the
highest score.
    classIds = []
    confidences = []
    boxes = []
    for out in outs:
        for detection in out:
            scores = detection[5:]
            classId = np.argmax(scores)
            confidence = scores[classId]
            if confidence > confThreshold:
                center_x = int(detection[0] * frameWidth)
                center_y = int(detection[1] * frameHeight)
                width = int(detection[2] * frameWidth)
                height = int(detection[3] * frameHeight)
                left = int(center_x - width / 2)
                top = int(center_y - height / 2)
                classIds.append(classId)
                confidences.append(float(confidence))
                boxes.append([left, top, width, height])

    # Perform non maximum suppression to eliminate redundant overlapping boxes with
    # lower confidences.
    indices = cv.dnn.NMSBoxes(boxes, confidences, confThreshold, nmsThreshold)
    for i in indices:
        i = i[0]
        box = boxes[i]
        left = box[0]
        top = box[1]
        width = box[2]
        height = box[3]
        drawPred(classIds[i], confidences[i], left, top, left + width, top + height)

# Process inputs
winName = 'Deep learning object detection in OpenCV'
cv.namedWindow(winName, cv.WINDOW_NORMAL)

outputFile = "yolo_out_py.avi"
if (args.image):
    # Open the image file
    if not os.path.isfile(args.image):
        print("Input image file ", args.image, " doesn't exist")
        sys.exit(1)
    cap = cv.VideoCapture(args.image)
    outputFile = args.image[:-4]+'_yolo_out_py.jpg'
elif (args.video):
    # Open the video file
    if not os.path.isfile(args.video):
        print("Input video file ", args.video, " doesn't exist")
        sys.exit(1)
    cap = cv.VideoCapture(args.video)
    outputFile = args.video[:-4]+'_yolo_out_py.avi'
else:
    # Webcam input
    cap = cv.VideoCapture(0)

# Get the video writer initialized to save the output video
if (not args.image):

```

```

vid_writer = cv.VideoWriter(outputFile, cv.VideoWriter_fourcc('M','J','P','G'), 30,
(round(cap.get(cv.CAP_PROP_FRAME_WIDTH)),round(cap.get(cv.CAP_PROP_FRAME_HEIGHT))))

while cv.waitKey(1) < 0:

    # get frame from the video
    hasFrame, frame = cap.read()

    # Stop the program if reached end of video
    if not hasFrame:
        print("Se h terminado el proceso !!!")
        print("El archivo se guarda en ", outputFile)
        cv.waitKey(3000)
        # Release device
        cap.release()
        break

    # Create a 4D blob from a frame.
    blob = cv.dnn.blobFromImage(frame, 1/255, (inpWidth, inpHeight), [0,0,0], 1, crop=False)

    # Sets the input to the network
    net.setInput(blob)

    # Runs the forward pass to get output of the output layers
    outs = net.forward(getOutputsNames(net))

    # Remove the bounding boxes with low confidence
    postprocess(frame, outs)

    # Put efficiency information. The function getPerfProfile returns the overall time for
    inference(t) and the timings for each of the layers(in layersTimes)
    t, _ = net.getPerfProfile()
    #La función devuelve el número de ticks por segundo. Es decir, el siguiente código calcula
    el tiempo de ejecución en segundos:
    label = 'Inference time: %.2f ms' % (t * 1000.0 / cv.getTickFrequency())
    cv.putText(frame, label, (0, 15), cv.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255))

    # Write the frame with the detection boxes
    if (args.image):
        cv.imwrite(outputFile, frame.astype(np.uint8))
    else:
        vid_writer.write(frame.astype(np.uint8))

    cv.imshow(winName, frame)

```

Recursos:

<https://youtu.be/cGmGOi2kkJ4>

<https://www.learnopencv.com/deep-learning-based-object-detection-using-yolov3-with-opencv-python-c/>

Versión 1: Prueba de funcionamiento YOLOv3

Versión 2: Entrenamiento para detección objeto en concreto.

Versión 3: Fusión con algoritmo SORT de seguimiento

Félix Calvo