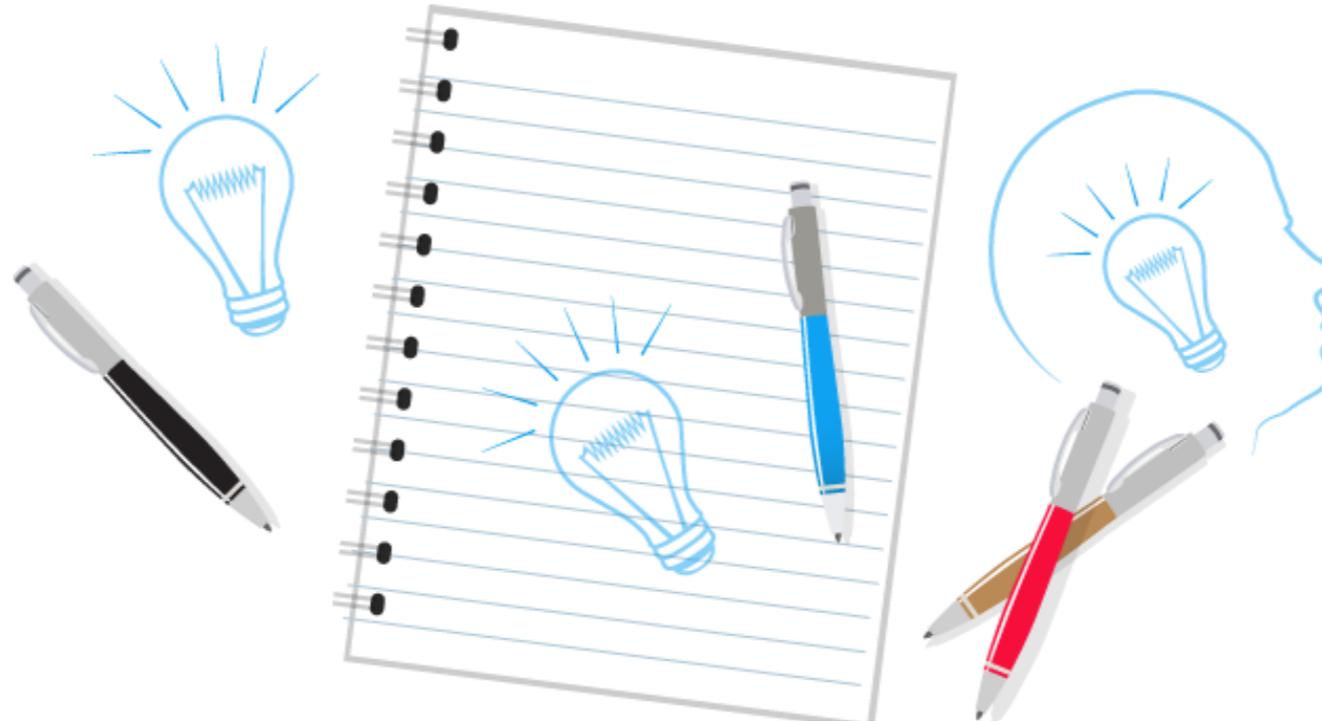


CAPÍTULO 11. Computer Vision

v.1.2 OCTUBRE 2024

Ricardo Moraleda Gareta

[Director departamento de software de GDO Software]





Computer Vision



Tensor Flow



Node-RED

Computer Vision

v.1.2 OCTUBRE 2024



Mobilenet



Hand pose



COCO SSD



PoseNet



Face Api



Open CV



C#





Visión Artificial



TensorFlow JavaScript

Para realizar proyectos de introducción a la visión artificial me basaré en la librería JS **TENSOR FLOW** y más concretamente en los nodos de Node-RED realizados por @kazuhiyokoi 

<https://www.tensorflow.org>

<https://github.com/kazuhiyokoi/node-red-contrib-tensorflow>

<https://kazuhiyokoi.medium.com/image-recognition-using-tensorflow-js-and-node-red-46e2df1b555e>

Definición: 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.

Entrenar modelos lleva mucho tiempo y TensorFlow dispone diferentes ya entrenados listos para utilizar.

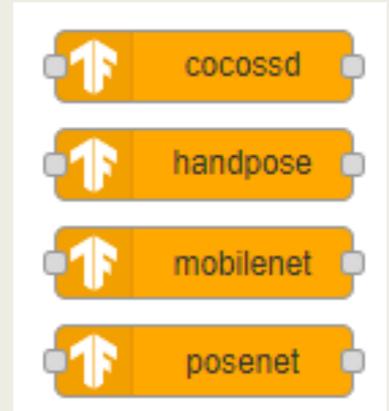
Para realizar ejemplos utilizaré Node-RED y en concreto el nodo "node-red-contrib-tensorflow".

Esta librería se basa en **4 modelos** (tratamiento imagen):

<https://www.tensorflow.org/js/models?hl=es>

<https://github.com/tensorflow/tfjs-models>

Type	Model	Demo	Details
Images	MobileNet	source	Classify images with labels from the ImageNet database.
	HandPose	live source	Real-time hand pose detection in the browser using TensorFlow.js.
	PoseNet	live source	A machine learning model which allows for real-time human pose estimation in the browser. See a detailed description here .
	Coco SSD	source	Object detection model that aims to localize and identify multiple objects in a single image. Based on the TensorFlow object detection API.



- Detección de objetos (COCO-SSD)
- Detección de la postura de la mano (HandPose)
- Clasificación de imágenes con ImageNet(MobileNet)
- Estimación de pose (PoseNet)



Visión Artificial



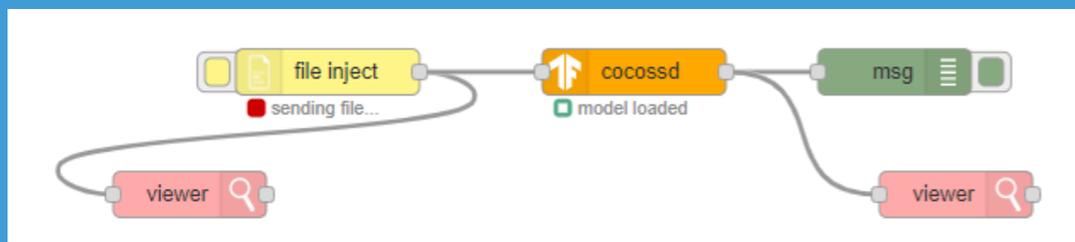
Detección de objetos (COCO-SSD)

Enlace con más info: <https://cocodataset.org/>

<https://towardsdatascience.com/understanding-ssd-multibox-real-time-object-detection-in-deep-learning-495ef744fab>

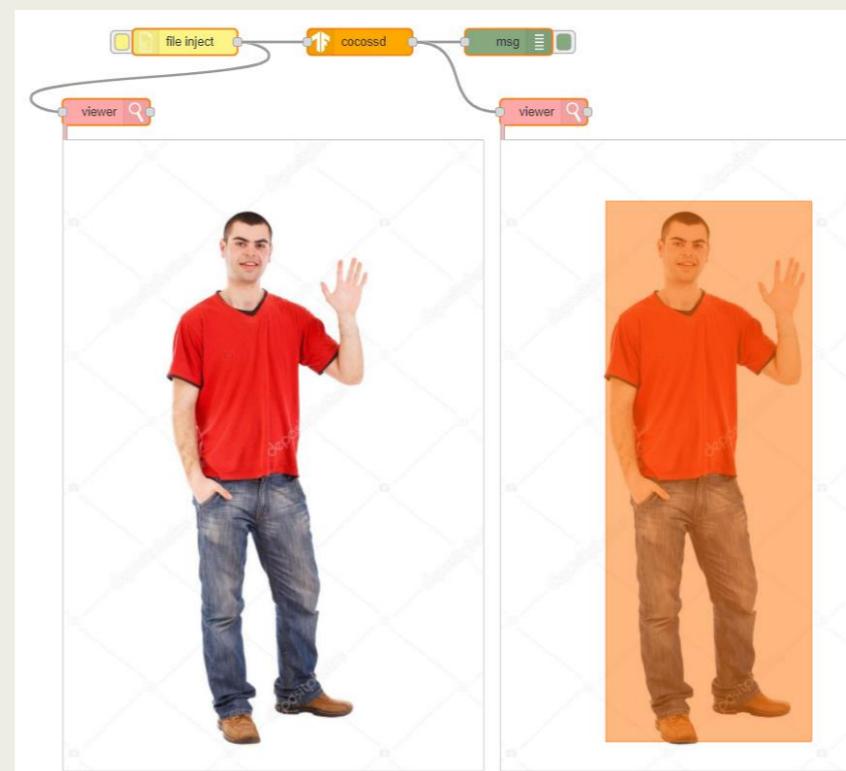
Common Objects in COntext - Single Shot MultiBox Detection. Es un conjunto de datos con miles de fotos clasificadas y categorizadas y es capaz de, dada una imagen, decirte qué es (entre 91 categorías) e incluso varias cosas en la misma imagen (multibox).

La librería te encuadra (bounding box) lo que ha detectado, qué clase es y un resultado (de 0 a 1) que es la probabilidad de que sea eso.



Node-RED (cocossd)

Resultado: "Person" con 99% prob.



```

msg : Object
  object
    payload: "person"
    _msgid: "978ce087.937b5"
  details: array[1]
    0: object
      bbox: array[4]
        0: 170.32321155071259
        1: 99.92831933498383
        2: 333.9137872457504
        3: 875.2063604593277
      class: "person"
      score: 0.9906831979751587
  annotatedInput: buffer[380276]
  
```



Visión Artificial



Clasificación de imágenes (MobileNet)

Node-RED (mobilenet)

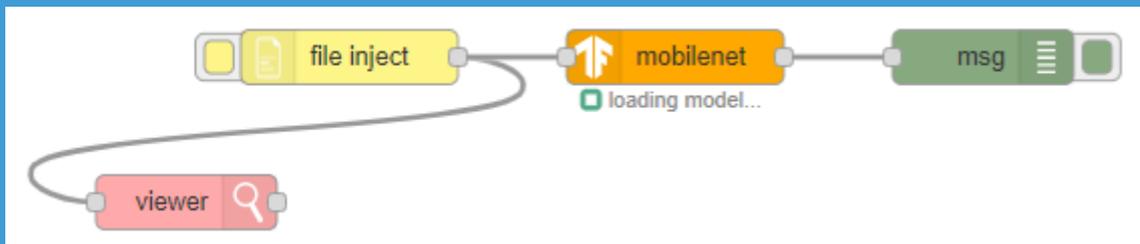
Enlace con más info:

https://github.com/tensorflow/models/blob/master/research/slim/nets/mobilenet_v1.md

Clasifica imágenes con etiquetas de la base de datos de ImageNet (MobileNet).

Inyectando la misma imagen de antes (el paisano con camiseta roja y jeans azules).

Resultado: "blue jean" con un 60 % prob.



```

msg : Object
  object
    payload: array[3]
      0: "jean"
      1: "blue jean"
      2: "denim"
    _msgid: "3c0b899b.3fd7a6"
    details: array[3]
      0: object
        className: "jean, blue jean, denim"
        probability: 0.6020798087120056
      1: object
        className: "poncho"
        probability: 0.061974186450242996
      2: object
        className: "sweatshirt"
        probability: 0.04622947424650192
  
```



Visión Artificial



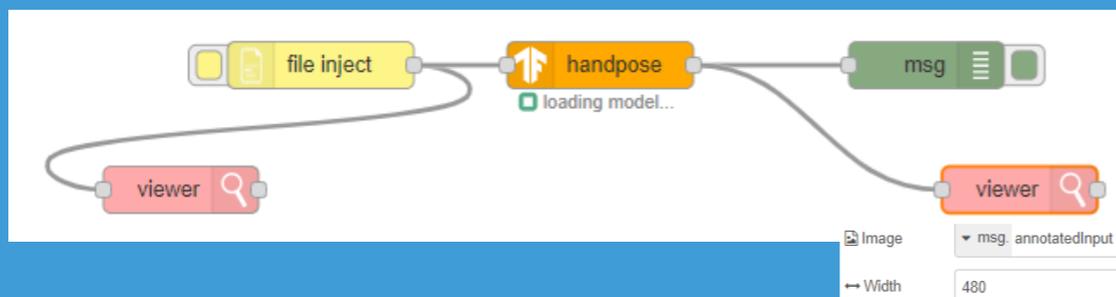
Detección postura de la mano (HandPose)

Enlace con más info:

<https://github.com/tensorflow/tfjs-models/tree/master/handpose>

Detecta la palma de la mano y seguimiento de los dedos a partir de los huesos de la mano. Predicción en 3D de 21 puntos clave de la mano por cada mano detectada.

Inyectando la misma imagen de antes (el paisano con camiseta roja y jeans azules).



Node-RED (handpose)

Resultado: Detecta la mano con un 99,98 % prob. y la posición de cada dedo respecto a la palma.

```

msg : Object
  object
    payload: object
      _msgid: "a97b79bc.0c9598"
    details: array[1]
      0: object
        handInViewConfidence:
          0.9998652935028076
        boundingBox: object
          topLeft: array[2]
          bottomRight: array[2]
        landmarks: array[21]
        annotations: object
          thumb: array[4]
          indexFinger: array[4]
          middleFinger: array[4]
          ringFinger: array[4]
          pinky: array[4]
          palmBase: array[1]
        annotatedInput: buffer[446731]
  
```



Visión Artificial



Estimación de pose (PoseNet)

Node-RED (posenet)

Enlace con más info:

<https://github.com/tensorflow/tfjs-models/tree/master/posenet>

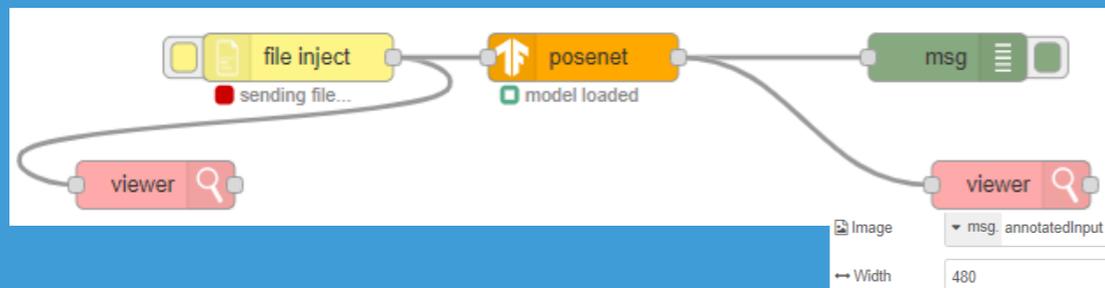
<https://medium.com/tensorflow/real-time-human-pose-estimation-in-the-browser-with-tensorflow-js-7dd0bc881cd5>

Para probar una demo: <https://storage.googleapis.com/tfjs-models/demos/posenet/camera.html>

Resultado: Detecta la pose con un 96,36 % prob. y la posición de cada parte del cuerpo.

Estima poses humanas en tiempo real.

Inyectando la misma imagen de antes (el paisano con camiseta roja y jeans azules).



A Node-RED flow diagram showing the result of pose estimation. It features a 'file inject' node connected to a 'posenet' node, which is connected to a 'msg' node. The 'msg' node is connected to two 'viewer' nodes. The left viewer shows a man in a red shirt and blue jeans. The right viewer shows the same man with orange lines overlaid on his body, representing the pose estimation. To the right of the viewers is a JSON object representing the output of the 'posenet' node.

```

object
  payload: object
    _msgid: "7d97a3de.1f8b8c"
  details: object
    score: 0.9636909260469324
  keypoints: array[17]
    [0 ... 9]
      0: object
        score: 0.9986575245857239
        part: "nose"
        position: object
      1: object
        score: 0.9959567785263062
        part: "leftEye"
        position: object
      2: object
        score: 0.9981282353401184
        part: "rightEye"
        position: object
      3: object
        score: 0.6655983924865723
        part: "leftEar"
        position: object
      4: object
      5: object
      6: object
      7: object
      8: object
      9: object
    [10 ... 16]
  annotatedInput: buffer[425660]
  
```



Visión Artificial

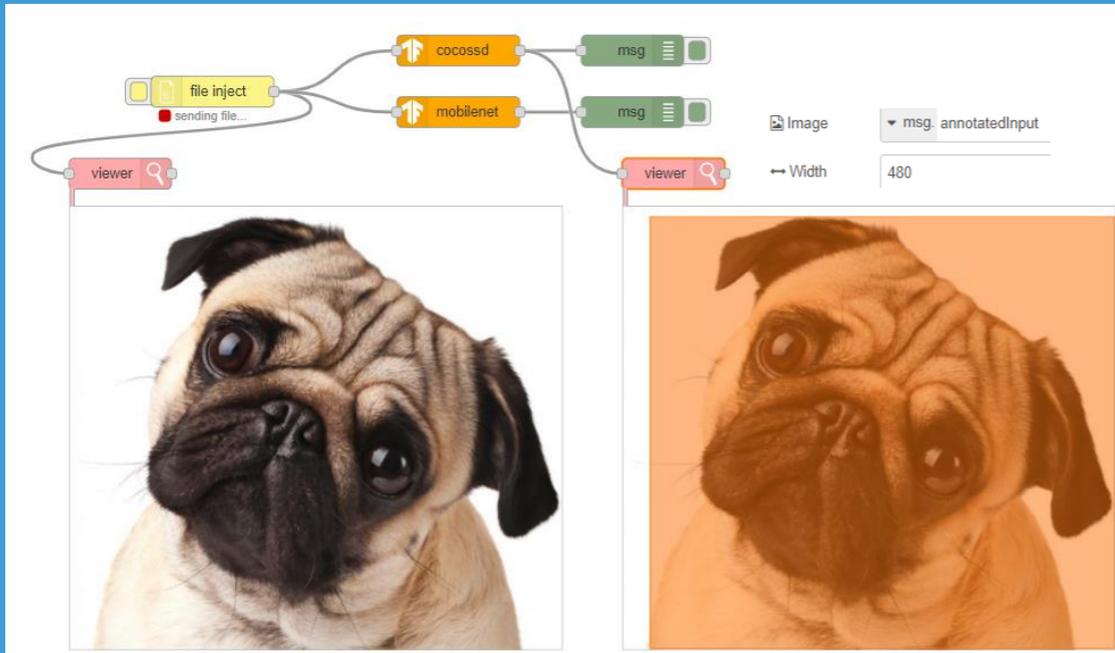


Cocossd+MobileNet

Cocossd+MobileNet

Más ejemplos:

Resultados:



```

msg : Object
  ▼ object
    ▶ payload: array[2]
      _msgid: "e1060143.613b5"
    ▼ details: array[3]
      ▼ 0: object
        className: "pug, pug-dog"
        probability: 0.9901100993156433
      ▼ 1: object
        className: "bull mastiff"
        probability: 0.009512982331216335
      ▼ 2: object
        className: "Brabancon griffon"
        probability: 0.0001959530054591596

msg : Object
  ▼ object
    payload: "dog"
    _msgid: "e1060143.613b5"
  ▼ details: array[1]
    ▼ 0: object
      ▶ bbox: array[4]
        class: "dog"
        score: 0.9153035283088684
    ▶ annotatedInput: buffer[355661]
  
```



Visión Artificial

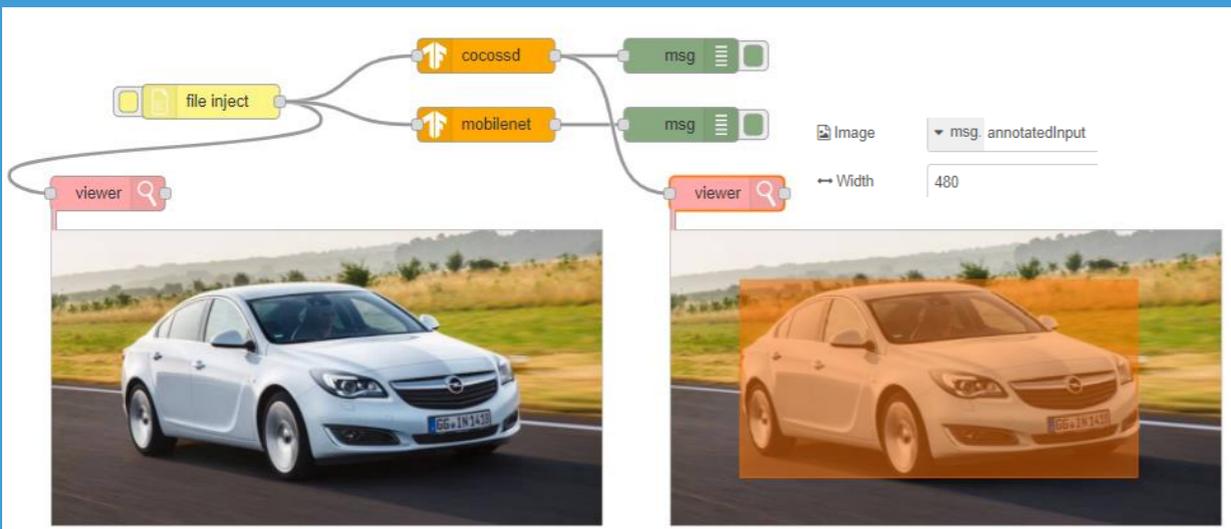


Cocossd+MobileNet

Cocossd+MobileNet

Más ejemplos:

Resultados:



```

msg : Object
  ▼ object
    ▼ payload: array[2]
      0: "sports car"
      1: "sport car"
      _msgid: "694f35c0.494d8c"
    ▼ details: array[3]
      ▼ 0: object
        className: "sports car, sport car"
        probability: 0.5514271259307861
      ▼ 1: object
        className: "racer, race car, racing car"
        probability: 0.14680978655815125
      ▼ 2: object
        className: "car wheel"
        probability: 0.07212120294570923
  msg : Object
    ▼ object
      payload: "car"
      _msgid: "694f35c0.494d8c"
    ▼ details: array[1]
      ▼ 0: object
        ▶ bbox: array[4]
        class: "car"
        score: 0.9571360945701599
    ▶ annotatedInput: buffer[289774]
  
```



Visión Artificial

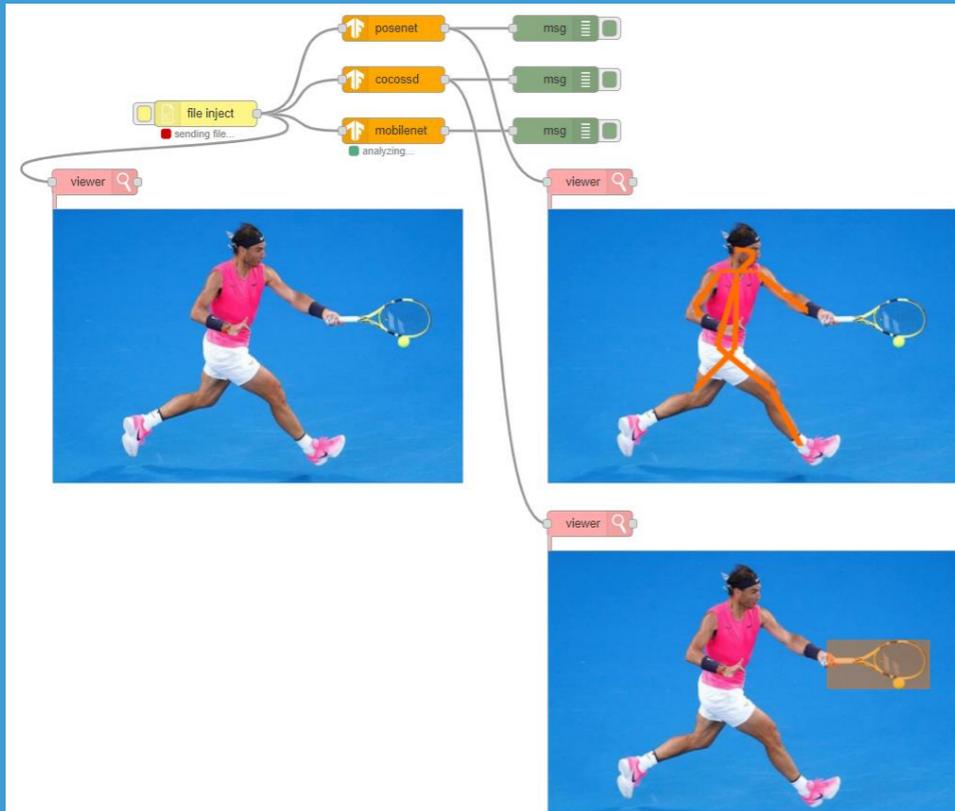


Cocossd+MobileNet+PoseNet

Cocossd+MobileNet+PoseNet

Más ejemplos:

Resultados:



```

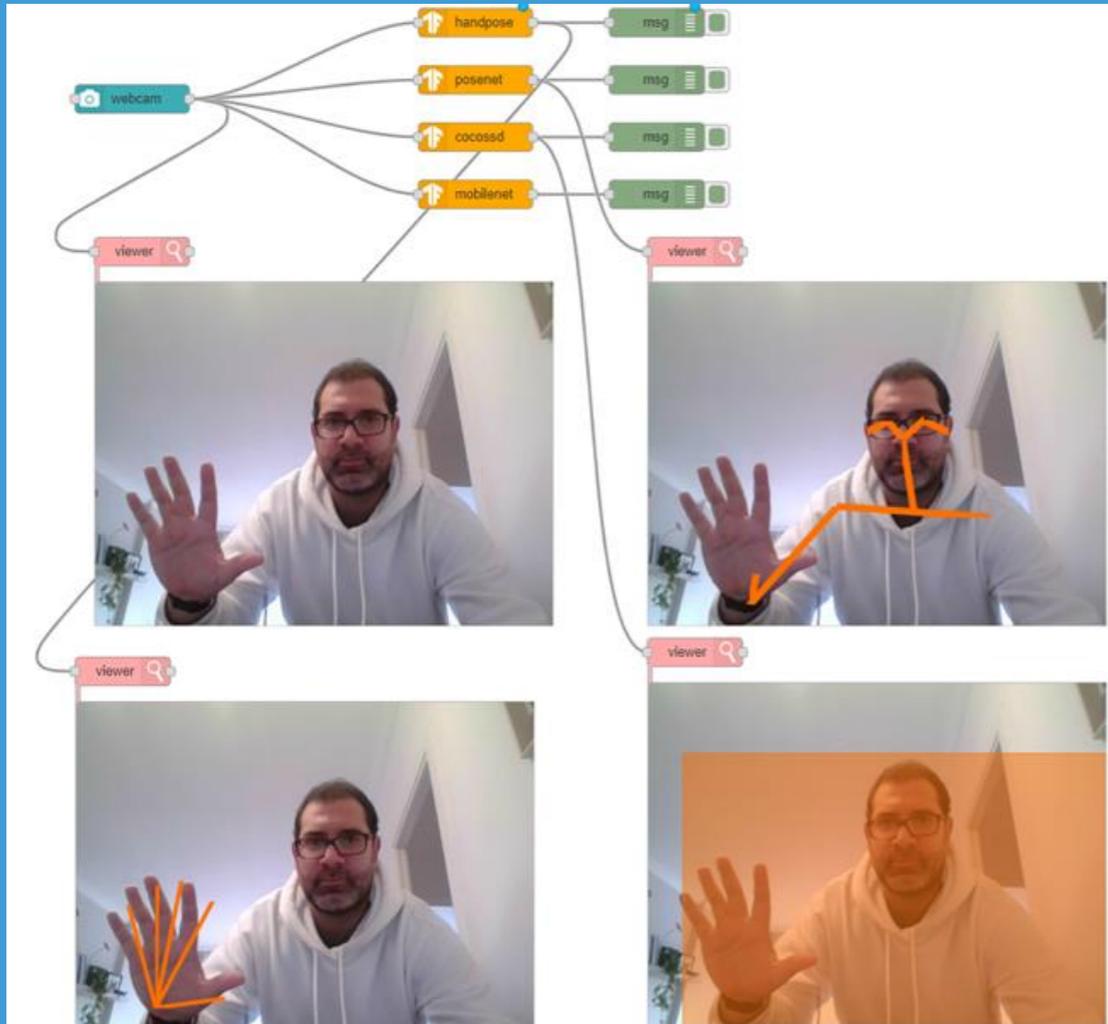
msg : Object
  object
    payload: array[2]
    _msgid: "2366bdf2.047672"
    details: array[3]
      0: object
        className: "racket, racquet"
        probability: 0.6790947318077087
      1: object
        className: "tennis ball"
        probability: 0.2901928424835205
      2: object
  
```

```

msg : Object
  object
    payload: object
      _msgid: "2366bdf2.047672"
    details: object
      score: 0.9079812063890345
      keypoints: array[17]
      annotatedInput: buffer[355024]
msg : Object
  object
    payload: "tennis racket"
    _msgid: "2366bdf2.047672"
    details: array[3]
      0: object
        bbox: array[4]
        class: "tennis racket"
        score: 0.9870955348014832
      1: object
        bbox: array[4]
        class: "person"
        score: 0.9754883050918579
      2: object
        bbox: array[4]
        class: "sports ball"
        score: 0.5165106654167175
    annotatedInput: buffer[355357]
  
```



Visión Artificial



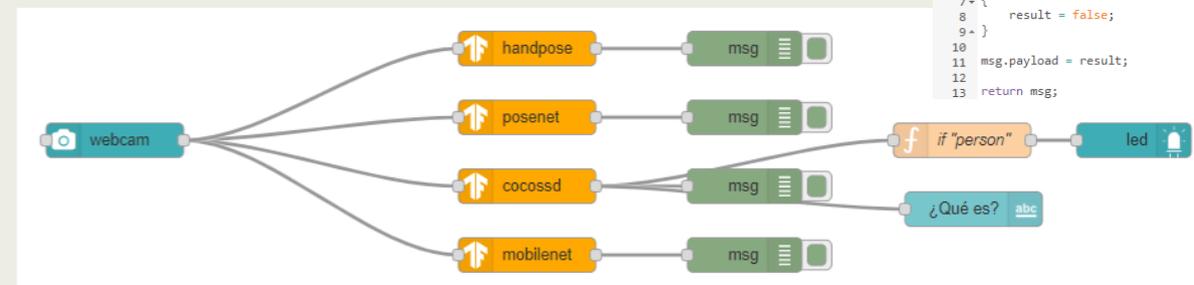
La imagen es obtenida por la cámara del portátil. **node_ui_webcam.** Ojo con los permisos del navegador!!



```

1 var result;
2
3- if (msg.payload == "person") {
4   result = true;
5- }
6 else
7- {
8   result = false;
9- }
10
11 msg.payload = result;
12
13 return msg;

```





Visión Artificial



Otra librería es: node-red-contrib-face-recognition (para el reconocimiento de caras humanas)

<https://flows.nodered.org/node/node-red-contrib-face-recognition>

<https://github.com/justadudewhohacks/face-api.js>

Face Recognition

Name: TBBFaces

Detection Type: SSD

Faces to Detect: Multiple Faces

Detection Confidence: 50

- Create child process
- Add Facial Landmarks to each face
- Predict Facial Expressions for each face
- Predict Age and Gender for each face
- Recognise a face in image

face api input

file inject

base64

Input Image

Find Faces

Debug

Set Payload to Image

Labeled Image

Detalle del resultado:

0.99

happy : 100%
female : 93.13%
46 years

0.99

happy : 100%
male : 98%
41 years



Visión Artificial



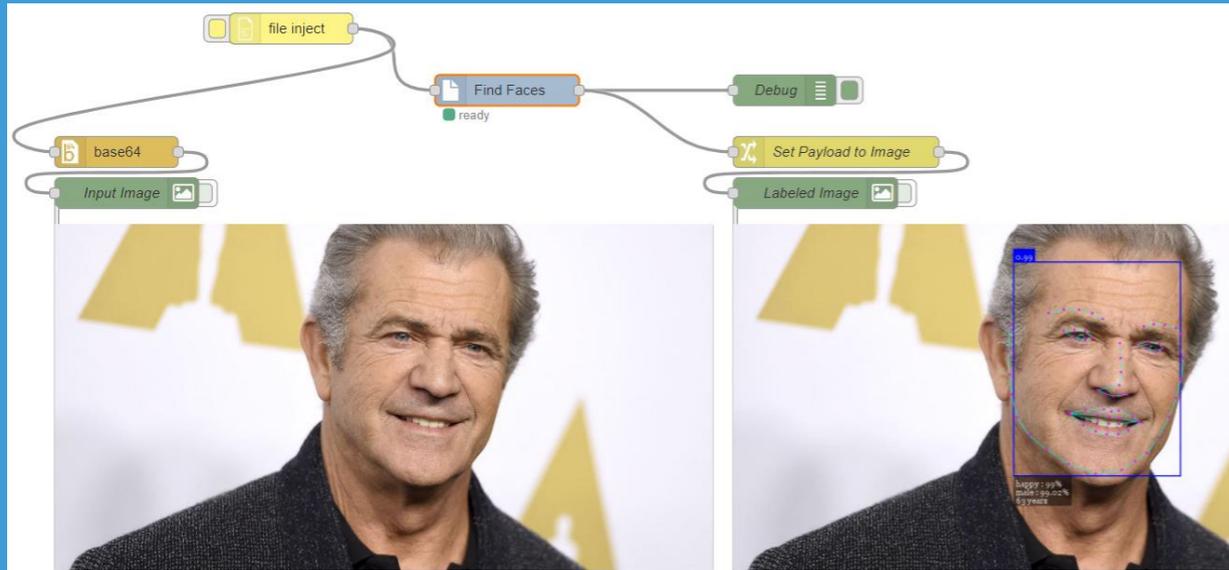
Otro ejemplo:

Detection confidence: 0.9

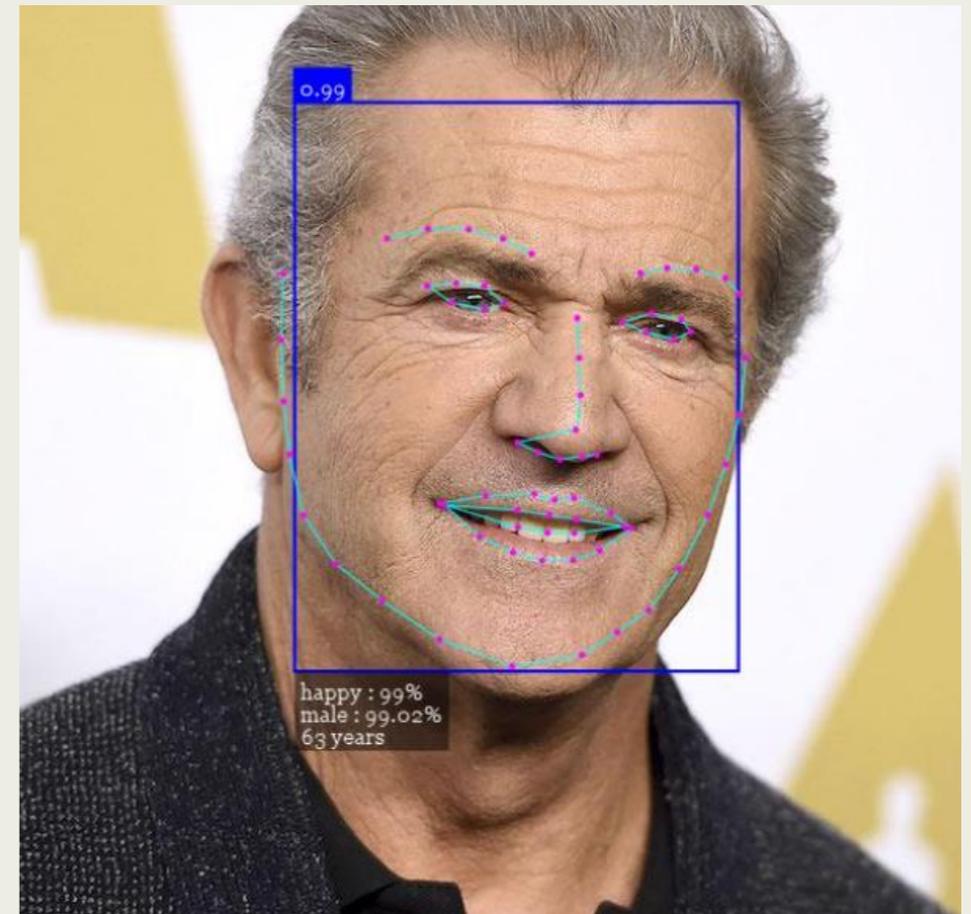
Happy: 99%

Male: 99.02%

63 years



Detalle del resultado:





Visión Artificial



Otra característica es el reconocimiento de una cara en una imagen.

Uso como patrón una foto mía (en un momento dado) y envío al sistema una nueva foto.

Name: TBBFaces

Detection Type: SSD

Faces to Detect: Multiple Faces

Detection Confidence: 50

Create child process

Add Facial Landmarks to each face

Predict Facial Expressions for each face

Predict Age and Gender for each face

Recognise a face in image

Recognition Metric: Mean Squared Error

Matched Confidence: 50

Add Descriptor: Add Images Remove Descriptors

1 descriptor exists for this node

Pattern (descriptor)



new input

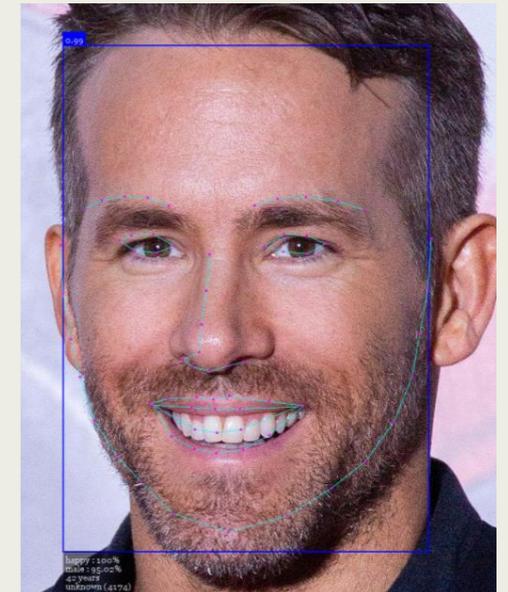
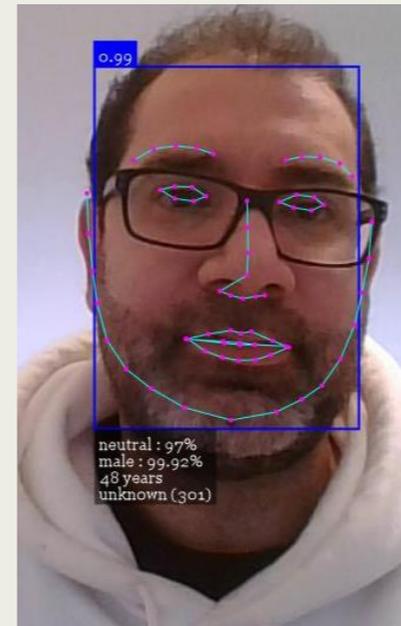


Detalle del resultado. La distancia va de igual (0) a totalmente diferente (10.000). Si la salida es < 1.300 (empíricamente) puedes decir que reconoce el patrón y puedes permitir desbloquear un acceso, por ejemplo.

```
if (msg.payload.TBBFaces.faces[0].matchedDistance <= 1300) { msg.payload = true;}
```

matchedDistance: 301
matchedMetric: "Mean Squared Error"

matchedDistance: 4174
matchedMetric: "Mean Squared Error"

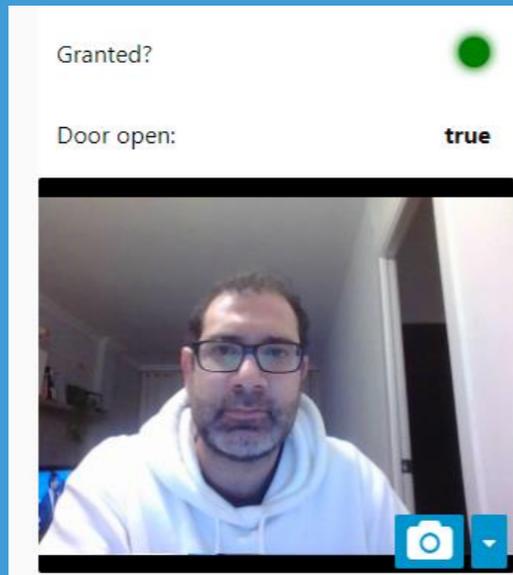
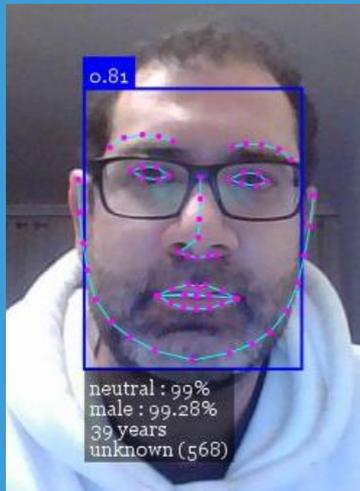
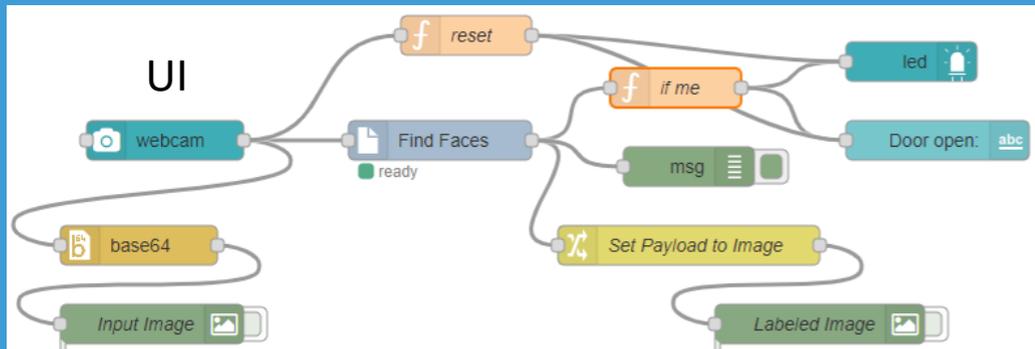




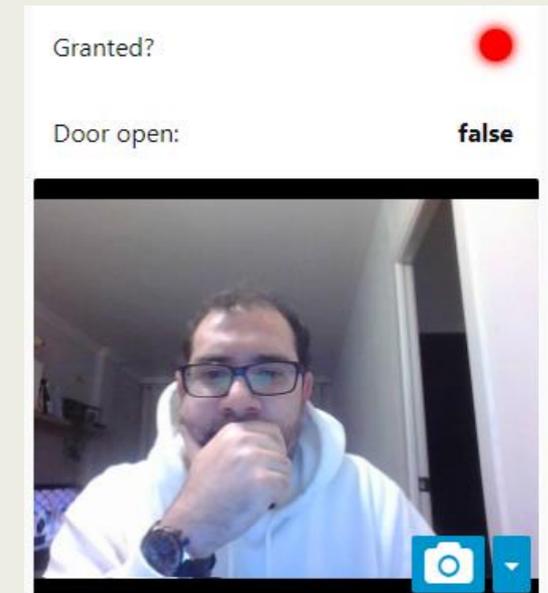
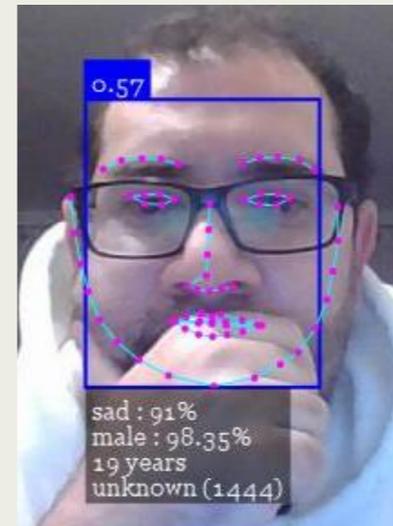
Visión Artificial



Imagen con node_ui_webcam. Control de acceso con el mismo patrón (descriptor) de antes (distancia: 568 → OK)



Distancia: 1444 → >1300 (KO). Al tener la mano en la boca pierde precisión para hacer el match. Además el detection confidence pasa de 0.81 a 0.57.



Sencillo control de acceso mediante visión artificial. Esto puede desenclavar una apertura de una puerta, incluso una sirena acústica. La cámara puede ser externa (IP) y la detección mediante un sensor de presencia para hacer la foto y el posterior cálculo (computing).



Visión Artificial

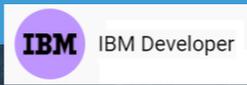


Node-RED + TensorFlow

Proyecto muy interesante para validar la entrada de una persona a una instalación si lleva mascarilla, casco, etc.

<https://www.youtube.com/watch?v=bOdIPwWej98>

@Paul Van Eck



Face Mask Detector Test

Utiliza otras librerías de TensorFlow. La primera entra en conflicto con estas.

https://developer.ibm.com/tutorials/building-a-machine-learning-node-for-node-red-using-tensorflowjs/?cm_mmc=OSocial_Blog_-_Developer_IBM+Developer_-_WW_WW_-_ibmdev-OfInfluencer-YouTube-KA-node-red-tensorflow&cm_mmca1=000037FD&cm_mmca2=10010797

En el vídeo está el enlace al código fuente (github).

Hard Hat Detector Test



Detección de Objetos



OpenCvSharp4

Un ejemplo simple de C# con OpenCVSharp (una biblioteca de OpenCV para .NET) que aplica un modelo YOLO v4 previamente entrenado basado en el conjunto de datos COCO (Common Objects in COntext) para la detección de objetos.

En este caso detecta coches (ni camiones, ni motos) y los cuenta.

https://www.linkedin.com/posts/ricardo-moraleda-gareta-9421099_opencvsharp-yolo4tiny-opencv-activity-7247972995065856000--xyR?utm_source=share&utm_medium=member_desktop



#	OpenCvSharp4 por shimat	4.10.0.20240616
#	OpenCV wrapper for .NET. Since this package includes only core managed libraries, another package of native bindings for your OS is required (OpenCvSharp4.runtime.*).	
#	OpenCvSharp4.runtime.win por shimat	4.10.0.20240616
#	Internal implementation package for OpenCvSharp to work on Windows except UWP.	

El programa desarrollado se basa en el modelo YOLOv4 (You Only Look Once).

Escanea frame a frame y con un umbral de confianza > del 90% se detectan coches y se cuentan.



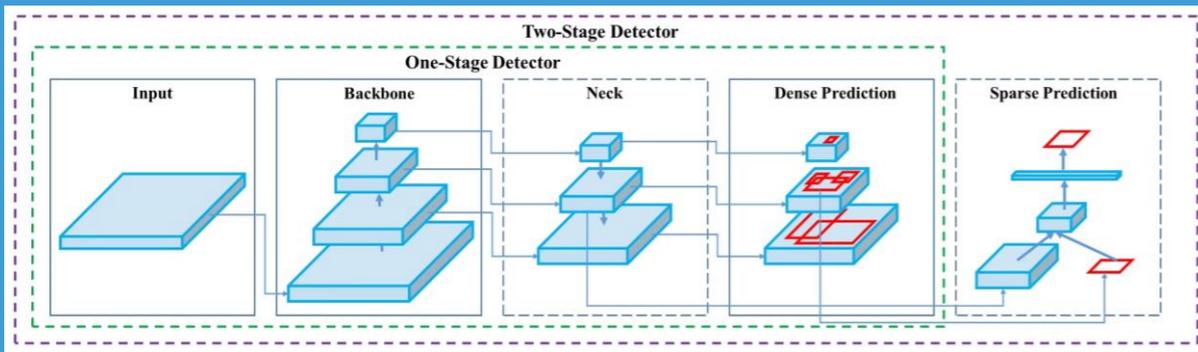


Detección de Objetos



YOLOv4

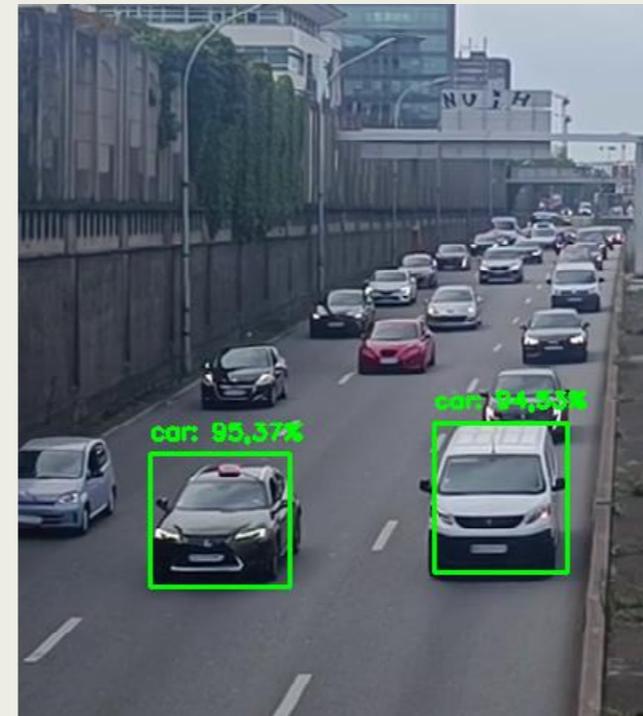
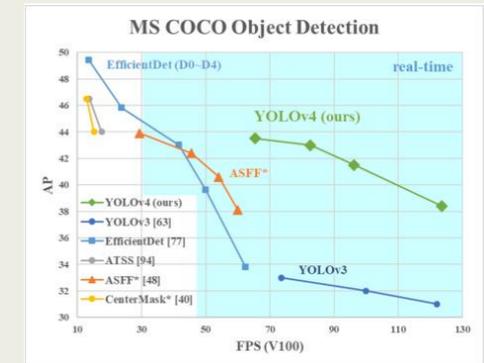
YOLOv4, un detector de objetos en tiempo real de última generación lanzado en 2020 por Alexey Bochkovskiy.



Un detector de objetos típico se compone de varias partes: la entrada, la columna vertebral, el cuello y la cabeza. La columna vertebral de YOLOv4 está preentrenada en ImageNet y se utiliza para predecir las clases y los cuadros delimitadores de los objetos. La columna vertebral puede proceder de varios modelos, como VGG, ResNet, ResNeXt o DenseNet. La parte del cuello del detector se utiliza para recoger mapas de características de distintas etapas y suele incluir varias trayectorias ascendentes y varias descendentes. La parte de la cabeza es la que se utiliza para hacer las detecciones y clasificaciones finales de los objetos.

El objeto al ser detectado se recuadra en verde y se indica el % de detección. Superior al umbral de confianza configurado.

<https://arxiv.org/pdf/2004.10934>



Computer Vision

v.1.2 OCTUBRE 2024



<https://www.linkedin.com/in/ricardo-moraleda-gareta-9421099>

<https://www.linkedin.com/company/gdo-electric1996/>

RICARDO MORALEDA GARETA