PARCIAL – PRACTICO

Leer TODOS los requerimientos para realizar el trabajo (ver objetos que puede detectar nuestro model).

1. Descargar imagenes de un data set público con un objeto de interes (ej. Flores, autos, etc.). VER qué objetos detecta el modelo a usar.

2. Creen un directorio con la siguiente estructura y de acuerdo al objeto que tienen:

data/

├── object\_labels.csv

├── object\_img

│ ├── 000001.jpg

│ ├── 000002.jpg

│ ├── ...

├── object\_img\_v1

│ ├── test

│ │ ├── Clase 2000

│ │ │ ├── 000046.jpg

│ │ │ ├── 000047.jpg

│ │ │ ├── ...

│ │ ├── Clase 2001

│ │ │ ├── 000450.jpg

│ │ │ ├── 000451.jpg

│ │ │ ├── ...

│ ├── train

│ │ ├── Clase 2000

│ │ │ ├── 000001.jpg

│ │ │ ├── 000002.jpg

│ │ │ ├── ...

"""

3. Completar la función para obtener las coordenadas de los objetos: *detectar\_coordenadas.py*

4. Completar la función *remover\_fondo\_imagen.py* llamando a la función anterior y recortando el fondo de la imagen.

5. Las imágenes sin el fondo deben quedar en un nuevo directorio con la misma estructura del directorio original.

CREAR FUNCIÓN *remover\_fondo\_imagen.py*

def main(folder\_data\_entrada, folder\_data\_salida):

"""

Parameters

----------

folder\_data\_entrada: str

Path absoluto al directorio train/test de las imagen.

folder\_data\_salida: str

Path absolute al directorio donde se guardaran las imagenes recortadas.

Esta función debe hacer lo siguiente:

1. Iterar sobre cada imagen del archivo ` folder\_data\_entrada `. Como consejo pueden usar la función Python `os.walk()`

2. Cargar la imagen

3. Correr el detector y obtener las coordenadas del objeto con la función de deteccion de coordenadas.

4. Extraer el objeto de la imagen y guardarla en el archivo ` folder\_data\_salida ` con el mismo nombre de la imagen.

5.

if \_\_name\_\_ == "\_\_main\_\_":

args = parse\_args()

main(args. folder\_data\_entrada, args.folder\_data\_salida)

CREAR FUNCIÓN *detectar\_coordenadas.py*

from detectron2 import model\_zoo

from detectron2.config import get\_cfg

from detectron2.engine.defaults import DefaultPredictor

# Investiguen sobre el model para detección: "COCO-Detection/faster\_rcnn\_R\_101\_FPN\_3x.yaml". Este modelo en particular tiene un buen balance entre velocidad y precisión.

#Vean el ejemplo para correr modelos de Detectron2 en el siguiente Colab:

#https://colab.research.google.com/drive/16jcaJoc6bCFAQ96jDe2HwtXj7BMD\_-m5.

# Van a tener que cargar el modelo de detección en la variable global DET\_MODEL

cfg = get\_cfg()

cfg.merge\_from\_file(

model\_zoo.get\_config\_file("COCO-Detection/faster\_rcnn\_R\_101\_FPN\_3x.yaml")

)

cfg.MODEL.ROI\_HEADS.SCORE\_THRESH\_TEST = 0.5

cfg.MODEL.WEIGHTS = model\_zoo.get\_checkpoint\_url(

"COCO-Detection/faster\_rcnn\_R\_101\_FPN\_3x.yaml"

)

try:

# Puede fallar si no encuentra GPU

DET\_MODEL = DefaultPredictor(cfg)

except:

# Cargar modelo para CPU solamente

print(

f"Failed to load Detection model on GPU, "

"trying with CPU. Message: {exp}."

)

cfg.MODEL.DEVICE='cpu'

DET\_MODEL = DefaultPredictor(cfg)

Def object\_coordinates(img):

"""

Esta función corre el detector (cargado en la variable DET\_MODEL variable) sobre la imagen que le pasen, y obtiene la posición del objeto en la imagen, devolviendo sus coordenadas.

Tener presente:

1. Ver qué objetos puede detectar nuestro modelo (hasta 80 objetos diferentes).

2. El detector puede encontrar más de un mismo objeto en la imagen. Deben elegir aquel con la mayor superficie.

3. El model puede fallar y detectar cero objetos en la imagen. En tal caso, deben devolver las coordenadas que cubren toda la imagen, ejemplo: [0, 0, width, height].

4. Las coordenadas tienen que ser expresadas como “enteros”. Como trabajamos valores de coordenadas, deben ser enteros para referenciar la posición en un numpy.array.

Parametros

----------

img : numpy.ndarray

Imagen formato RGB .

Returns

-------

coordenadas: list

[x1, y1, x2, y2]

"""

coordenadas = None

return coordenadas